Heaps and Balanced Trees

For in-class use only in CSC 311 course

Dr. Marek A. Suchenek ©

April 28, 2014

1 Heaps and Balanced Trees

1.1 Binary representations of positive natural numbers

This section has already been presented in class. It is included here for students' convenience.

How many bits are needed to represent a number $M > 0$ in binary?

Say, it's n.

The greatest binary number that may be represented on n bits is n 1s.

The greatest binary number that may be represented on n bits is n 1s.

$$2^{n-1} = 1\underbrace{00\ldots0}_{n-1} = \underbrace{100\ldots0}_{n} \leq M \leq \underbrace{11\ldots1}_{n}.$$

The greatest binary number that may be represented on n bits is n 1s.

$$2^{n-1} = 1\underbrace{00\ldots0}_{n-1} = \underbrace{100\ldots0}_{n} \leq M \leq \underbrace{11\ldots1}_{n}.$$

$$2^{n-1} \leq M \leq \underbrace{11\ldots1}_{n}.$$

The greatest binary number that may be represented on n bits is n 1s.

$$2^{n-1} \leq M \leq \underbrace{11\ldots1}_{n}.$$

$$\underbrace{11\ldots1}_{n}+1 = 1\,\underbrace{00\ldots0}_{n} = 2^n.$$

$$\underbrace{11\ldots1}_{n}+1 = 1\underbrace{00\ldots0}_{n} = 2^n.$$

$$\underbrace{11\ldots1}_{n} = 2^n - 1.$$

$$\underbrace{11\ldots1}_{n} = 2^n - 1.$$

$$2^{n-1} \leq M \leq 2^n - 1$$

$$2^{n-1} \leq M \leq 2^n - 1$$

$$2^{n-1} \leq M < 2^n$$

$$2^{n-1} \leq M < 2^n$$

$$\log_2 2^{n-1} \leq \log_2 M < \log_2 2^n$$

$$\log_2 2^{n-1} \leq \log_2 M < \log_2 2^n$$

$$n - 1 \leq \log_2 M < n.$$

$$n - 1 \leq \log_2 M < n.$$

$$n - 1 = \lfloor \log_2 M \rfloor$$

$$n - 1 = \lfloor \log_2 M \rfloor$$

$$n = \lfloor \log_2 M \rfloor + 1.$$

$$n = \lfloor \log_2 M \rfloor + 1.$$

So, $\lfloor \log_2 M \rfloor + 1$ bits are needed to represent number $M > 0$ in binary.

Here is an important equality that I recommend you try to memorize:

$$\lfloor \log_2 M \rfloor + 1 = \lceil \log_2 (M+1) \rceil,$$

for any integer $M \geq 1$.

1.2 Heaps

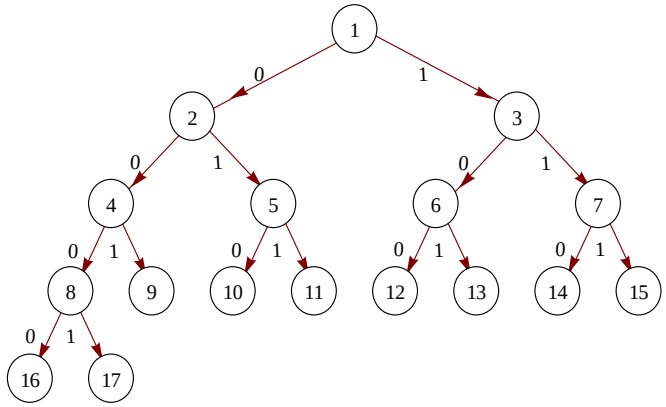


Figure 1: A heap with 17 nodes showing nodes' ordinal numbers in decimal.

Figure 2: A really large heap on a small picture.

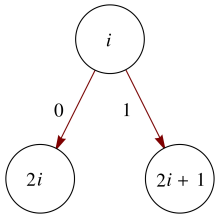


Figure 3: Ancestral information translated onto ordinal numbers.

$$j = \begin{cases} 2i \text{ if } j \text{ is the left child of } i \\ \\ 2i + 1 \text{ if } j \text{ is the right child of } i. \end{cases}$$

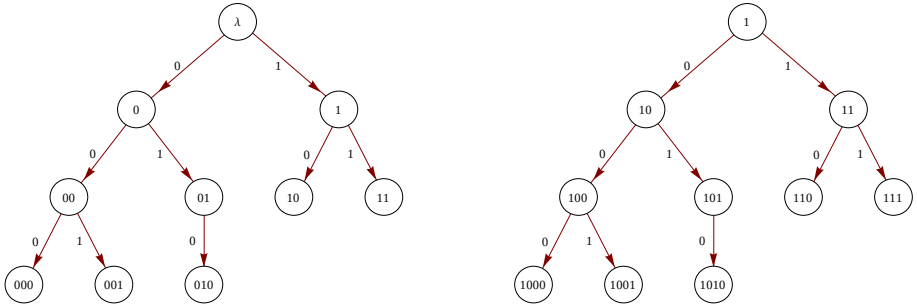


Figure 4: Two visualizations of a heap with 10 nodes: on the left showing binary sequences that represent the nodes (with λ being the empty sequence), on the right showing their ordinal numbers in binary, and edges' labels on both showing suffixes (the last digit that makes the difference between a child and its parent).

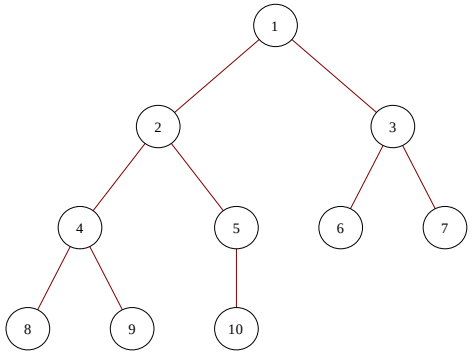


Figure 5: A heap with 10 nodes showing their ordinal numbers in decimal.

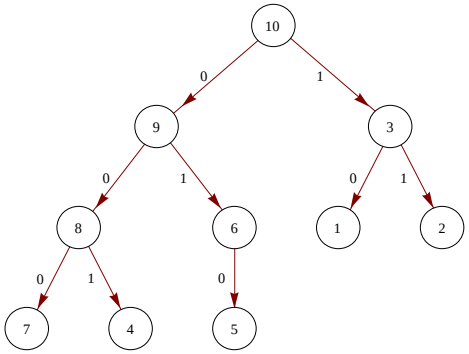


Figure 6: A heap with 10 nodes showing their values and not the ordinal numbers.

1	2	3	4	5	6	7	8	9	10
10	9	3	8	6	1	2	7	4	5

Figure 7: Array representation of the heap of Figure 6.

.

1.3 The height (or depth) of a heap

The depth D_n of the heap with n nodes is equal to the length of the longest path from its root to any other node.

The depth D_n of the heap with n nodes is equal to the length of the longest path from its root to any other node.

The path from the root to the node n is a longest such path.

The depth D_n of the heap with n nodes is equal to the length of the longest path from its root to any other node.

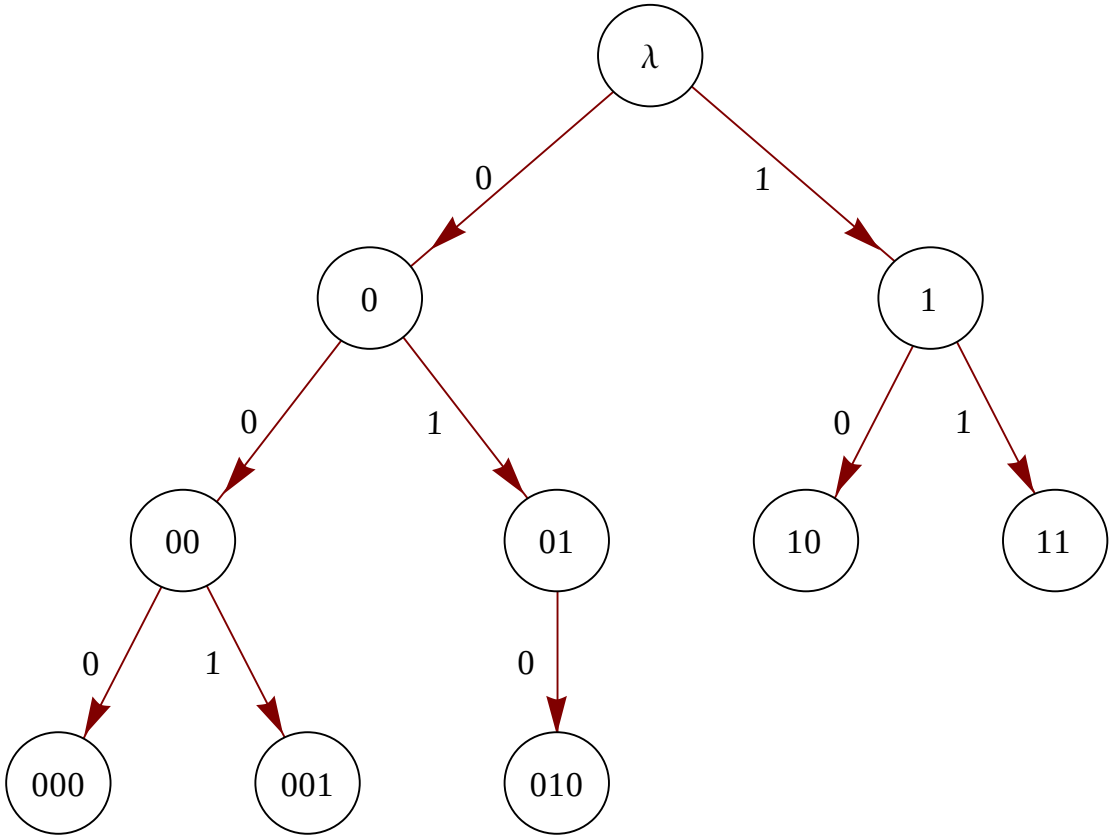
The path from the root to the node n is a longest such path.

So, the depth D_n of the heap with n nodes is equal to the length of the longest path from its root to the node n.

So, the depth D_n of the heap with n nodes is equal to the length of the longest path from its root to the node n.

So, the depth D_n of the heap with n nodes is equal to the length of the longest path from its root to the node n.

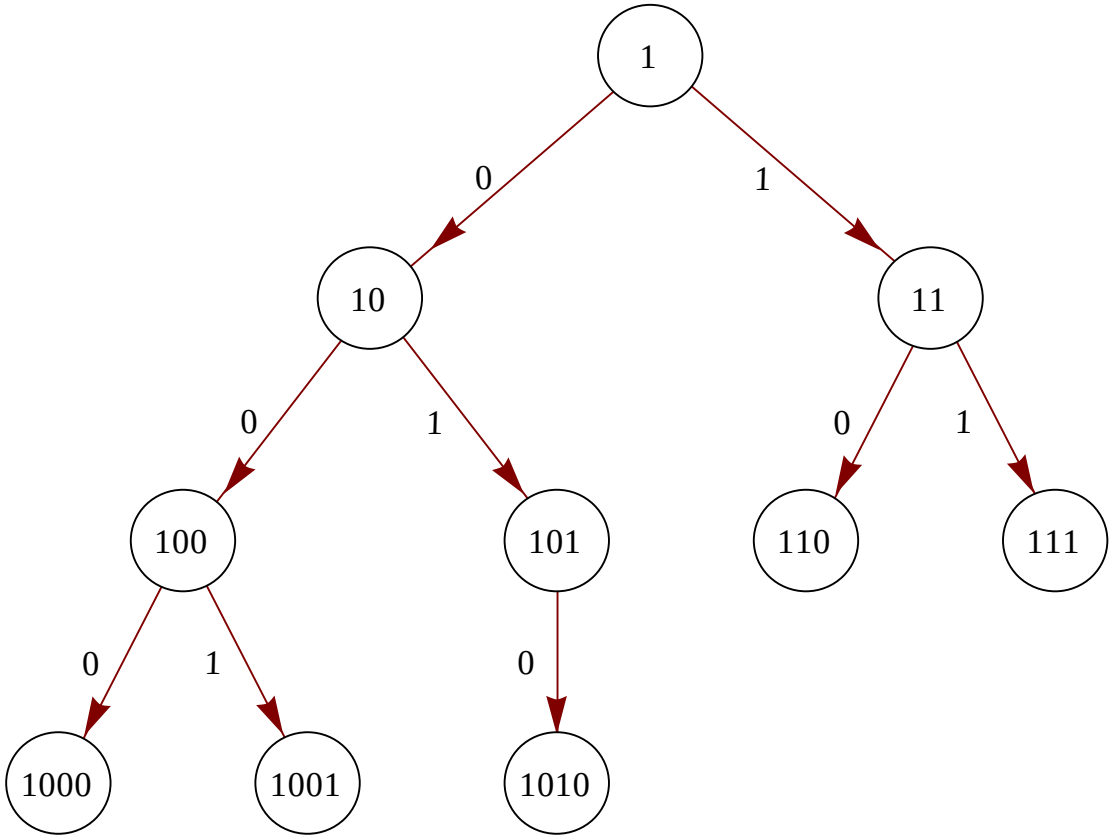
It is the length of the binary sequence that comprises the node n.


λ
0
1
0
1
0
1
0
1
00
01
10
11
0
1
0
000
001
010

So, the depth D_n of the heap with n nodes is equal to the number of bits in the binary sequence that comprises the node n.

So, the depth D_n of the heap with n nodes is equal to the number of bits in the binary sequence that comprises the node n.

And that is one less than the number of bits needed to represent n.


1
0
1
10
11
0
1
0
1
100
101
110
111
0
1
0
1000
1001
1010

The depth D_n of the heap with n nodes is one less than the number of bits needed to represent n.

The depth D_n of the heap with n nodes is one less than the number of bits needed to represent n.

$$D_n = (\lfloor \log_2 n \rfloor + 1) - 1 = \lfloor \log_2 n \rfloor .$$

The depth D_n of the heap with n nodes is one less than the number of bits needed to represent n.

$$D_n = \lfloor \log_2 n \rfloor .$$

1.4 The running time of H.remove() and H.insert(x)

Let H be a heap with n nodes.

The number of comparisons $C_{remove}(n)$ done in the worst case by H.remove()

Let H be a heap with n nodes.

The number of comparisons $C_{remove}(n)$ done in the worst case by H.remove()

$$2 \times D_{n-1} - 1 \leq C_{remove}(n) \leq 2 \times D_{n-1},$$

$$2 \times D_{n-1} - 1 \leq C_{remove}(n) \leq 2 \times D_{n-1},$$

$$2 \times \lfloor \log_2(n-1) \rfloor - 1 \leq C_{remove}(n),$$

$$2 \times D_{n-1} - 1 \leq C_{remove}(n) \leq 2 \times D_{n-1},$$

$$C_{remove}(n) \leq 2 \times \lfloor \log_2(n-1) \rfloor,$$

$$2 \times \lfloor \log_2(n-1) \rfloor - 1 \leq C_{remove}(n),$$

$$C_{remove}(n) \leq 2 \times \lfloor \log_2(n-1) \rfloor,$$

$$2 \times \lfloor \log_2(n-1) \rfloor - 1 \leq C_{remove}(n),$$

$$C_{remove}(n) \leq 2 \times \lfloor \log_2(n-1) \rfloor,$$

$$C_{remove}(n) \in \Theta(\log n).$$

$$C_{remove}(n) \in \Theta(\log n).$$

The number of comparisons $C_{insert}(n)$ done in the worst case by H.insert(x)

The number of comparisons $C_{insert}(n)$ done in the worst case by H.insert(x)

$$C_{insert}(n) = D_{n+1} = \lfloor \log_2(n+1) \rfloor .$$

The number of comparisons $C_{insert}(n)$ done in the worst case by H.insert(x)

$$C_{insert}(n) = \lfloor \log_2(n+1) \rfloor .$$

$$C_{insert}(n) = \lfloor \log_2(n+1) \rfloor.$$

$$C_{insert}(n) \in \Theta(\log n).$$

$$C_{insert}(n) \in \Theta(\log n).$$

1.5 **The running time of** PriorityQueue-Sort**, a.k.a.** HeapSort

```
73      public static void  PriorityQueueSort(int[] A)
74      {
75          int n = A.length;
76          PriorityQueue H = new PriorityQueue();
77          for (int i = 0; i < n; i++) H.insert(A[i]);
78          for (int i = n-1; i >= 0; i--) A[i]=H.remove();
79      }
```

Figure 8: A simple HeapSort.

$$C_{sort}(n) \in O(n \times \max(C_{insert}(n), C_{remove}(n)).$$

$$C_{sort}(n) \in O(n \times \max(C_{insert}(n), C_{remove}(n)).$$

$$C_{sort}(n) \in O(n \times \max(\log n, \log n).$$

$$C_{sort}(n) \in O(n \times \max(\log n, \log n).$$

$$C_{sort}(n) \in O(n \log n).$$

1.6 A more accurate computation of the running time of PriorityQueueSort, a.k.a. HeapSort

```
73      public static void  PriorityQueueSort(int[] A)
74      {
75          int n = A.length;
76          PriorityQueue H = new PriorityQueue();
77          for (int i = 0; i < n; i++) H.insert(A[i]);
78          for (int i = n-1; i >= 0; i--) A[i]=H.remove();
79      }
```

Figure 9: A simple HeapSort.

Assume that the array to be sorted has n elements. The worst-case number of comparisons in the first for-loop is

Assume that the array to be sorted has n elements. The worst-case number of comparisons in the first for-loop is

```
73      public static void  PriorityQueueSort(int[] A)
74      {
75          int n = A.length;
76          PriorityQueue H = new PriorityQueue();
77          for (int i = 0; i < n; i++) H.insert(A[i]);
78          for (int i = n-1; i >= 0; i--) A[i]=H.remove();
79      }
```

```
73      public static void  PriorityQueueSort(int[] A)
74      {
75          int n = A.length;
76          PriorityQueue H = new PriorityQueue();
77          for (int i = 0; i < n; i++) H.insert(A[i]);
78          for (int i = n-1; i >= 0; i--) A[i]=H.remove();
79      }
```

$$\Sigma_{i=0}^{n-1} C_{insert}(i)$$

Assume that the array to be sorted has n elements. The worst-case number of comparisons in the first for-loop is

$$\Sigma_{i=0}^{n-1} C_{insert}(i) = \Sigma_{i=0}^{n-1} \lfloor \log_2(i+1) \rfloor$$

$$\Sigma_{i=0}^{n-1} C_{insert}(i) = \Sigma_{i=0}^{n-1} \lfloor \log_2(i+1) \rfloor$$

$$\Sigma_{i=0}^{n-1} C_{insert}(i) = \Sigma_{i=1}^{n} \lfloor \log_2 i \rfloor$$

$$\Sigma_{i=0}^{n-1} C_{insert}(i) = \Sigma_{i=1}^{n} \lfloor \log_2 i \rfloor$$

$$\Sigma_{i=0}^{n-1} C_{insert}(i) \leq n \lfloor \log_2 n \rfloor .$$

$$\Sigma_{i=0}^{n-1} C_{insert}(i) = \Sigma_{i=1}^{n} \lfloor \log_2 i \rfloor$$

$$\Sigma_{i=0}^{n-1} C_{insert}(i) \leq n \lfloor \log_2 n \rfloor.$$

because $\lfloor \log_2 i \rfloor \leq \lfloor \log_2 n \rfloor$ and there are n terms to add in $\Sigma_{i=1}^{n} \lfloor \log_2 i \rfloor$.

The number of comparisons in the second for-loop is

The number of comparisons in the second for-loop is

```
73      public static void  PriorityQueueSort(int[] A)
74      {
75          int n = A.length;
76          PriorityQueue H = new PriorityQueue();
77          for (int i = 0; i < n; i++) H.insert(A[i]);
78          for (int i = n-1; i >= 0; i--) A[i]=H.remove();
79      }
```

```
73      public static void  PriorityQueueSort(int[] A)
74      {
75          int n = A.length;
76          PriorityQueue H = new PriorityQueue();
77          for (int i = 0; i < n; i++) H.insert(A[i]);
78          for (int i = n-1; i >= 0; i--) A[i]=H.remove();
79      }
```

$$\Sigma_{i=0}^{n-1} C_{remove}(i+1)$$

$$\Sigma_{i=0}^{n-1} C_{remove}(i+1)$$

$$\Sigma_{k=1}^{n} C_{remove}(k), \text{ where } k = i+1$$

$$\Sigma_{k=1}^{n} C_{remove}(k)$$

$$\Sigma_{k=2}^{n} C_{remove}(k)$$

$$\Sigma_{k=2}^{n} C_{remove}(k)$$

$$\Sigma_{i=2}^{n} C_{remove}(i)$$

$$\Sigma_{i=2}^{n} C_{remove}(i) \leq \Sigma_{i=2}^{n} 2 \times \lfloor \log_2(i-1) \rfloor$$

$$\Sigma_{i=2}^{n} C_{remove}(i) \leq \Sigma_{i=2}^{n} 2 \times \lfloor \log_2(i-1) \rfloor$$

$$\Sigma_{i=2}^{n} C_{remove}(i) \leq \Sigma_{i=2}^{n} 2 \times \lfloor \log_2 n \rfloor = 2(n-1) \lfloor \log_2 n \rfloor.$$

$$\Sigma_{i=2}^{n} C_{remove}(i) \leq \Sigma_{i=2}^{n} 2 \times \lfloor \log_2(i-1) \rfloor$$

$$\Sigma_{i=2}^{n} C_{remove}(i) \leq \Sigma_{i=2}^{n} 2 \times \lfloor \log_2 n \rfloor = 2(n-1) \lfloor \log_2 n \rfloor.$$

because $\lfloor \log_2(i-1) \rfloor \leq \lfloor \log_2 n \rfloor$ and
there are $n-1$ terms to add in $\Sigma_{i=2}^{n-1} \lfloor \log_2 i \rfloor$.

$$\Sigma_{i=2}^{n} C_{remove}(i) \leq 2(n-1)\lfloor \log_2 n \rfloor.$$

$$\Sigma_{i=2}^{n} C_{remove}(i) \leq 2(n-1)\lfloor \log_2 n \rfloor.$$

The worst-case number of comparisons in the first for-loop was estimated as:

$$\Sigma_{i=0}^{n-1} C_{insert}(i) \leq n\lfloor \log_2 n \rfloor.$$

Adding them together, we obtain the number $C_{sort}(n)$ of comparisons performed in both loops

```
73      public static void  PriorityQueueSort(int[] A)
74      {
75          int n = A.length;
76          PriorityQueue H = new PriorityQueue();
77          for (int i = 0; i < n; i++) H.insert(A[i]);
78          for (int i = n-1; i >= 0; i--) A[i]=H.remove();
79      }
```

Adding them together, we obtain the number $C_{sort}(n)$ of comparisons performed in both loops

$$C_{sort}(n) = \Sigma_{i=0}^{n-1} C_{insert}(i) + \Sigma_{i=2}^{n} C_{remove}(i)$$

Adding them together, we obtain the number $C_{sort}(n)$ of comparisons performed in both loops

$$C_{sort}(n) = \Sigma_{i=0}^{n-1} C_{insert}(i) + \Sigma_{i=2}^{n} C_{remove}(i) \leq$$
$$\leq n\lfloor \log_2 n \rfloor + 2(n-1)\lfloor \log_2 n \rfloor.$$

Adding them together, we obtain the number $C_{sort}(n)$ of comparisons performed in both loops

$$C_{sort}(n) \leq n\lfloor \log_2 n \rfloor + 2(n-1)\lfloor \log_2 n \rfloor .$$

Adding them together, we obtain the number $C_{sort}(n)$ of comparisons performed in both loops

$$C_{sort}(n) \leq n\lfloor \log_2 n \rfloor + 2(n-1)\lfloor \log_2 n \rfloor.$$

$$C_{sort}(n) \leq (3n-2)\lfloor \log_2 n \rfloor.$$

Note

It is known that the first for-loop may be replaced by a heap-construction program that performs the same task as the first for-loop while performing no more than $2n - 2$ comparisons.

This allows to decrease the number of comparisons while sorting down to no more than:

$$C_{sort}(n) \leq 2n - 2 + 2(n-1)\lfloor \log_2 n \rfloor.$$

This allows to decrease the number of comparisons while sorting down to no more than:

$$C_{sort}(n) \leq 2n - 2 + 2(n-1)\lfloor \log_2 n \rfloor.$$

$$C_{sort}(n) \leq 2(n-1)(\lfloor \log_2 n \rfloor + 1).$$

This allows to decrease the number of comparisons while sorting down to no more than:

$$C_{sort}(n) \leq 2n - 2 + 2(n-1)\lfloor \log_2 n \rfloor.$$

$$C_{sort}(n) \leq 2(n-1)(\lfloor \log_2 n \rfloor + 1).$$

Recall that $\lfloor \log_2 n \rfloor + 1$ is the number of bits necessary to represent n in binary.

How interesting!

1.7 Even more accurate computation of the running time of PriorityQueueSort, a.k.a. HeapSort

```
73      public static void  PriorityQueueSort(int[] A)
74      {
75          int n = A.length;
76          PriorityQueue H = new PriorityQueue();
77          for (int i = 0; i < n; i++) H.insert(A[i]);
78          for (int i = n-1; i >= 0; i--) A[i]=H.remove();
79      }
```

Figure 10: A simple HeapSort.

Assume that the array to be sorted has n elements. The number of comparisons in the first for-loop is

Assume that the array to be sorted has n elements. The number of comparisons in the first for-loop is

```
73      public static void  PriorityQueueSort(int[] A)
74      {
75          int n = A.length;
76          PriorityQueue H = new PriorityQueue();
77          for (int i = 0; i < n; i++) H.insert(A[i]);
78          for (int i = n-1; i >= 0; i--) A[i]=H.remove();
79      }
```

```
73      public static void  PriorityQueueSort(int[] A)
74      {
75          int n = A.length;
76          PriorityQueue H = new PriorityQueue();
77          for (int i = 0; i < n; i++) H.insert(A[i]);
78          for (int i = n-1; i >= 0; i--) A[i]=H.remove();
79      }
```

$$\Sigma_{i=0}^{n-1} C_{insert}(i)$$

Assume that the array to be sorted has n elements. The number of comparisons in the first for-loop is

$$\Sigma_{i=0}^{n-1} C_{insert}(i) = \Sigma_{i=0}^{n-1} \lfloor \log_2(i+1) \rfloor$$

$$\Sigma_{i=0}^{n-1} C_{insert}(i) = \Sigma_{i=0}^{n-1} \lfloor \log_2(i+1) \rfloor$$

$$\Sigma_{i=0}^{n-1} C_{insert}(i) = \Sigma_{i=1}^{n} \lfloor \log_2 i \rfloor$$

$$\Sigma_{i=0}^{n-1} C_{insert}(i) = \Sigma_{i=1}^{n} \lfloor \log_2 i \rfloor$$

$$\Sigma_{i=0}^{n-1} C_{insert}(i) \leq \Sigma_{i=1}^{n} \lfloor \log_2 i \rfloor + \lfloor \log_2 n \rfloor .$$

$$\Sigma_{i=0}^{n-1} C_{insert}(i) = \Sigma_{i=1}^{n-1} \lfloor \log_2 i \rfloor + \lfloor \log_2 n \rfloor .$$

$$\Sigma_{i=0}^{n-1} C_{insert}(i) = \Sigma_{i=2}^{n} \lfloor \log_2 (i-1) \rfloor + \lfloor \log_2 n \rfloor .$$

The number of comparisons in the first for-loop is

$$\Sigma_{i=0}^{n-1} C_{insert}(i) = \Sigma_{i=2}^{n} \lfloor \log_2(i-1) \rfloor + \lfloor \log_2 n \rfloor .$$

The number of comparisons in the first for-loop is

$$\Sigma_{i=2}^{n} \lfloor \log_2(i-1) \rfloor + \lfloor \log_2 n \rfloor.$$

The number of comparisons in the second for-loop is

The number of comparisons in the second for-loop is

```
73      public static void  PriorityQueueSort(int[] A)
74      {
75          int n = A.length;
76          PriorityQueue H = new PriorityQueue();
77          for (int i = 0; i < n; i++) H.insert(A[i]);
78          for (int i = n-1; i >= 0; i--) A[i]=H.remove();
79      }
```

```
73      public static void  PriorityQueueSort(int[] A)
74      {
75          int n = A.length;
76          PriorityQueue H = new PriorityQueue();
77          for (int i = 0; i < n; i++) H.insert(A[i]);
78          for (int i = n-1; i >= 0; i--) A[i]=H.remove();
79      }
```

$$\Sigma_{i=0}^{n-1} C_{remove}(i+1)$$

$$\Sigma_{i=0}^{n-1} C_{remove}(i+1)$$

$$\Sigma_{k=1}^{n} C_{remove}(k), \text{ where } k = i+1$$

$$\Sigma_{k=1}^{n} C_{remove}(k)$$

$$\Sigma_{k=2}^{n} C_{remove}(k)$$

$$\Sigma_{k=2}^{n} C_{remove}(k)$$

$$\Sigma_{i=2}^{n} C_{remove}(i)$$

$$\Sigma_{i=2}^{n}(2\times\lfloor\log_2(i-1)\rfloor-1) \leq \Sigma_{i=2}^{n}C_{remove}(i)$$

$$\Sigma_{i=2}^{n}C_{remove}(i) \leq \Sigma_{i=2}^{n}2\times\lfloor\log_2(i-1)\rfloor$$

$$2\times\Sigma_{i=2}^{n}\lfloor\log_2(i-1)\rfloor-(n-1)\leq\Sigma_{i=2}^{n}C_{remove}(i)$$

$$\Sigma_{i=2}^{n}C_{remove}(i)\leq 2\times\Sigma_{i=2}^{n}\lfloor\log_2(i-1)\rfloor$$

Adding them together, we obtain the number $C_{sort}(n)$ of comparisons performed in both loops

```
73      public static void  PriorityQueueSort(int[] A)
74      {
75          int n = A.length;
76          PriorityQueue H = new PriorityQueue();
77          for (int i = 0; i < n; i++) H.insert(A[i]);
78          for (int i = n-1; i >= 0; i--) A[i]=H.remove();
79      }
```

Adding them together, we obtain the number $C_{sort}(n)$ of comparisons performed in both loops

$$\Sigma_{i=2}^{n} \lfloor \log_2(i-1) \rfloor + \lfloor \log_2 n \rfloor +$$

$$+2 \times \Sigma_{i=2}^{n} \lfloor \log_2(i-1) \rfloor - (n-1) \leq C_{sort}(n)$$

$$\Sigma_{i=2}^{n}\lfloor\log_2(i-1)\rfloor+\lfloor\log_2 n\rfloor+$$

$$+2\times\Sigma_{i=2}^{n}\lfloor\log_2(i-1)\rfloor-(n-1)\leq C_{sort}(n)$$

$$3\times\Sigma_{i=2}^{n}\lfloor\log_2(i-1)\rfloor-n+\lfloor\log_2 n\rfloor+1\leq C_{sort}(n)$$

Adding them together, we obtain the number $C_{sort}(n)$ of comparisons performed in both loops

$$C_{sort}(n) \leq \Sigma_{i=2}^{n} \lfloor \log_2(i-1) \rfloor + \lfloor \log_2 n \rfloor +$$

$$+2 \times \Sigma_{i=2}^{n} \lfloor \log_2(i-1) \rfloor$$

$$C_{sort}(n) \leq \Sigma_{i=2}^{n} \lfloor \log_2(i-1) \rfloor + \lfloor \log_2 n \rfloor +$$

$$+2 \times \Sigma_{i=2}^{n} \lfloor \log_2(i-1) \rfloor$$

$$C_{sort}(n) \leq 3 \times \Sigma_{i=2}^{n} \lfloor \log_2(i-1) \rfloor + \lfloor \log_2 n \rfloor .$$

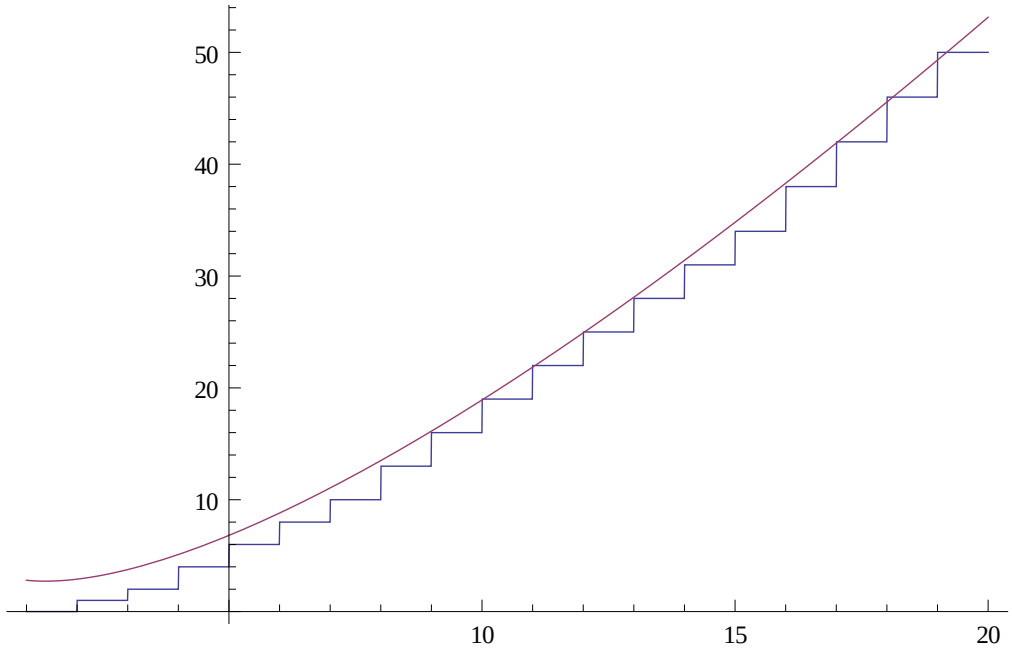
$$3 \times \Sigma_{i=2}^{n} \lfloor \log_2(i-1) \rfloor - n + \lfloor \log_2 n \rfloor + 1 \leq$$

$$\leq C_{sort}(n) \leq 3 \times \Sigma_{i=2}^{n} \lfloor \log_2(i-1) \rfloor + \lfloor \log_2 n \rfloor .$$

Let's compute

$$\Sigma_{i=2}^{n} \lfloor \log_2(i-1) \rfloor$$

$$\Sigma_{i=2}^{n} \lfloor \log_2(i-1) \rfloor \approx n \lg n - 1.9n + 4.7$$



Here is how to compute the exact value of

$$\Sigma_{i=2}^{n} \lfloor \log_2(i-1) \rfloor$$

$$\Sigma_{i=2}^{17}\lfloor \log_2 i \rfloor =$$
$$= \Sigma_{i=1}^{\lfloor \log_2 17 \rfloor - 1} i \times 2^i + (17 - 2^{\lfloor \log_2 17 \rfloor} + 1)\lfloor \log_2 17 \rfloor$$

$$\Sigma_{i=2}^{17}\lfloor\log_2 i\rfloor =$$
$$= \Sigma_{i=1}^{\lfloor\log_2 17\rfloor-1} i\times 2^i + (17-2^{\lfloor\log_2 17\rfloor}+1)\lfloor\log_2 17\rfloor$$

In general,

$$\Sigma_{i=2}^{n}\lfloor\log_2 i\rfloor =$$
$$= \Sigma_{i=1}^{\lfloor\log_2 n\rfloor-1} i\times 2^i + (n-2^{\lfloor\log_2 n\rfloor}+1)\lfloor\log_2 n\rfloor$$

In general,

$$\Sigma_{i=2}^{n} \lfloor \log_2 i \rfloor =$$
$$= \Sigma_{i=1}^{\lfloor \log_2 n \rfloor - 1} i \times 2^i + (n - 2^{\lfloor \log_2 n \rfloor} + 1) \lfloor \log_2 n \rfloor$$

$$\Sigma_{i=1}^{M} i \times 2^i = (M-1) \times 2^{M+1} + 2$$

$$\Sigma_{i=1}^{M} i \times 2^i = (M-1) \times 2^{M+1} + 2$$

$$\Sigma_{i=1}^{\lfloor \log_2 n \rfloor - 1} i \times 2^i =$$
$$= (\lfloor \log_2 n \rfloor - 1 - 1) \times 2^{\lfloor \log_2 n \rfloor - 1 + 1} + 2$$

$$\Sigma_{i=1}^{\lfloor \log_2 n \rfloor - 1} i \times 2^i =$$
$$= (\lfloor \log_2 n \rfloor - 1 - 1) \times 2^{\lfloor \log_2 n \rfloor - 1 + 1} + 2$$

$$\Sigma_{i=1}^{\lfloor \log_2 n \rfloor - 1} i \times 2^i =$$
$$= (\lfloor \log_2 n \rfloor - 2) \times 2^{\lfloor \log_2 n \rfloor} + 2$$

$$\Sigma_{i=1}^{\lfloor \log_2 n \rfloor - 1} i \times 2^i =$$
$$= (\lfloor \log_2 n \rfloor - 2) \times 2^{\lfloor \log_2 n \rfloor} + 2$$

$$\Sigma_{i=1}^{\lfloor \log_2 n \rfloor - 1} i \times 2^i =$$
$$= \lfloor \log_2 n \rfloor \times 2^{\lfloor \log_2 n \rfloor} - 2 \times 2^{\lfloor \log_2 n \rfloor} + 2$$

$$\Sigma_{i=1}^{\lfloor \log_2 n \rfloor - 1} i \times 2^i =$$

$$= \lfloor \log_2 n \rfloor \times 2^{\lfloor \log_2 n \rfloor} - 2 \times 2^{\lfloor \log_2 n \rfloor} + 2$$

We had:

$$\Sigma_{i=2}^{n} \lfloor \log_2 i \rfloor =$$

$$= \Sigma_{i=1}^{\lfloor \log_2 n \rfloor - 1} i \times 2^i + (n - 2^{\lfloor \log_2 n \rfloor} + 1) \lfloor \log_2 n \rfloor$$

$$\Sigma_{i=1}^{\lfloor \log_2 n \rfloor - 1} i \times 2^i =$$

$$= \lfloor \log_2 n \rfloor \times 2^{\lfloor \log_2 n \rfloor} - 2 \times 2^{\lfloor \log_2 n \rfloor} + 2$$

We had:

$$\Sigma_{i=2}^{n} \lfloor \log_2 i \rfloor =$$

$$= \Sigma_{i=1}^{\lfloor \log_2 n \rfloor - 1} i \times 2^i + (n - 2^{\lfloor \log_2 n \rfloor} + 1) \lfloor \log_2 n \rfloor$$

$$\Sigma_{i=2}^{n} \lfloor \log_2 i \rfloor =$$

$$= \lfloor \log_2 n \rfloor \times 2^{\lfloor \log_2 n \rfloor} - 2 \times 2^{\lfloor \log_2 n \rfloor} + 2 +$$

$$+(n - 2^{\lfloor \log_2 n \rfloor} + 1)\lfloor \log_2 n \rfloor$$

$$\Sigma_{i=2}^{n}\lfloor\log_2 i\rfloor =$$
$$= \lfloor\log_2 n\rfloor\times 2^{\lfloor\log_2 n\rfloor} - 2\times 2^{\lfloor\log_2 n\rfloor} + 2 +$$
$$+(n - 2^{\lfloor\log_2 n\rfloor} + 1)\lfloor\log_2 n\rfloor$$

$$\Sigma_{i=2}^{n}\lfloor\log_2 i\rfloor =$$
$$= \lfloor\log_2 n\rfloor\times 2^{\lfloor\log_2 n\rfloor} - 2^{\lfloor\log_2 n\rfloor+1} + 2 +$$
$$+n\lfloor\log_2 n\rfloor - \lfloor\log_2 n\rfloor\times 2^{\lfloor\log_2 n\rfloor} + \lfloor\log_2 n\rfloor$$

$$\Sigma_{i=2}^{n}\lfloor\log_2 i\rfloor =$$
$$= \lfloor\log_2 n\rfloor \times 2^{\lfloor\log_2 n\rfloor} - 2^{\lfloor\log_2 n\rfloor+1} + 2 +$$
$$+ n\lfloor\log_2 n\rfloor - \lfloor\log_2 n\rfloor \times 2^{\lfloor\log_2 n\rfloor} + \lfloor\log_2 n\rfloor$$

$$\Sigma_{i=2}^{n}\lfloor\log_2 i\rfloor =$$
$$= -2^{\lfloor\log_2 n\rfloor+1} + 2 + n\lfloor\log_2 n\rfloor + \lfloor\log_2 n\rfloor$$

$$\Sigma_{i=2}^{n}\lfloor\log_2 i\rfloor =$$
$$= -2^{\lfloor\log_2 n\rfloor+1}+2+n\lfloor\log_2 n\rfloor+\lfloor\log_2 n\rfloor$$

$$\Sigma_{i=2}^{n}\lfloor\log_2 i\rfloor =$$
$$= (n+1)\lfloor\log_2 n\rfloor - 2^{\lfloor\log_2 n\rfloor+1} + 2$$

$$\Sigma_{i=2}^{n} \lfloor \log_2 i \rfloor =$$
$$= (n+1)\lfloor \log_2 n \rfloor - 2^{\lfloor \log_2 n \rfloor + 1} + 2$$

$$\Sigma_{i=2}^{n} \lfloor \log_2 i \rfloor =$$
$$= (n+1)\lfloor \log_2 n \rfloor - 2^{\lfloor \log_2 n \rfloor + 1} + 2$$

$$\Sigma_{i=2}^{n} \lfloor \log_2 (i-1) \rfloor = \Sigma_{i=1}^{n-1} \lfloor \log_2 i \rfloor =$$

$$\Sigma_{i=2}^{n}\lfloor\log_2 i\rfloor =$$
$$= (n+1)\lfloor\log_2 n\rfloor - 2^{\lfloor\log_2 n\rfloor+1} + 2$$

$$\Sigma_{i=2}^{n}\lfloor\log_2(i-1)\rfloor = \Sigma_{i=1}^{n-1}\lfloor\log_2 i\rfloor =$$
$$\Sigma_{i=2}^{n}\lfloor\log_2 i\rfloor - \lfloor\log_2 n\rfloor =$$

$$\Sigma_{i=2}^{n} \lfloor \log_2 i \rfloor =$$
$$= (n+1)\lfloor \log_2 n \rfloor - 2^{\lfloor \log_2 n \rfloor + 1} + 2$$

$$\Sigma_{i=2}^{n} \lfloor \log_2 (i-1) \rfloor = \Sigma_{i=1}^{n-1} \lfloor \log_2 i \rfloor =$$
$$\Sigma_{i=2}^{n} \lfloor \log_2 i \rfloor - \lfloor \log_2 n \rfloor =$$
$$= n\lfloor \log_2 n \rfloor - 2^{\lfloor \log_2 n \rfloor + 1} + 2$$

$$\Sigma_{i=2}^{n}\lfloor\log_2(i-1)\rfloor = \Sigma_{i=1}^{n-1}\lfloor\log_2 i\rfloor =$$
$$\Sigma_{i=2}^{n}\lfloor\log_2 i\rfloor - \lfloor\log_2 n\rfloor =$$
$$= n\lfloor\log_2 n\rfloor - 2^{\lfloor\log_2 n\rfloor+1} + 2$$

$$\Sigma_{i=2}^{n}\lfloor\log_2(i-1)\rfloor =$$
$$= n\lfloor\log_2 n\rfloor - 2^{\lfloor\log_2 n\rfloor+1} + 2$$

$$\Sigma_{i=2}^{n}\lfloor\log_2(i-1)\rfloor =$$
$$= n\lfloor\log_2 n\rfloor - 2^{\lfloor\log_2 n\rfloor+1} + 2$$

We had:

$$3\times\Sigma_{i=2}^{n}\lfloor\log_2(i-1)\rfloor - n + \lfloor\log_2 n\rfloor + 1 \leq$$
$$\leq C_{sort}(n) \leq 3\times\Sigma_{i=2}^{n}\lfloor\log_2(i-1)\rfloor + \lfloor\log_2 n\rfloor.$$

$$3\times(n\lfloor\log_2 n\rfloor - 2^{\lfloor\log_2 n\rfloor+1} + 2) - n + \lfloor\log_2 n\rfloor + 1 \leq$$

$$\leq C_{sort}(n) \leq$$
$$\leq 3 \times (n\lfloor \log_2 n \rfloor - 2^{\lfloor \log_2 n \rfloor + 1} + 2) + \lfloor \log_2 n \rfloor .$$

$$3\times(n\lfloor\log_2 n\rfloor-2^{\lfloor\log_2 n\rfloor+1}+2)-n+\lfloor\log_2 n\rfloor+1\leq$$
$$\leq C_{sort}(n)\leq$$
$$\leq 3\times(n\lfloor\log_2 n\rfloor-2^{\lfloor\log_2 n\rfloor+1}+2)+\lfloor\log_2 n\rfloor.$$

$$(3n+1)\lfloor\log_2 n\rfloor-3\times 2^{\lfloor\log_2 n\rfloor+1}+6-n+1\leq$$
$$\leq C_{sort}(n)\leq$$
$$\leq (3n+1)\lfloor\log_2 n\rfloor-3\times 2^{\lfloor\log_2 n\rfloor+1}+6.$$

$$(3n+1)\lfloor \log_2 n \rfloor - 3 \times 2^{\lfloor \log_2 n \rfloor + 1} + 6 - n + 1 \leq$$
$$\leq C_{sort}(n) \leq$$
$$\leq (3n+1)\lfloor \log_2 n \rfloor - 3 \times 2^{\lfloor \log_2 n \rfloor + 1} + 6.$$

$$(3n+1)\lfloor \log_2 n \rfloor - 3 \times 2n + 6 - n + 1 \leq$$
$$\leq C_{sort}(n) <$$
$$< (3n+1)\lfloor \log_2 n \rfloor - 3 \times 2\frac{n}{2} + 6.$$

$$(3n+1)\lfloor \log_2 n \rfloor - 3 \times 2n + 6 - n + 1 \leq$$
$$\leq C_{sort}(n) <$$
$$< (3n+1)\lfloor \log_2 n \rfloor - 3 \times 2\frac{n}{2} + 6.$$

$$(3n+1)\lfloor \log_2 n \rfloor - 7n + 7 \leq$$
$$\leq C_{sort}(n) <$$
$$< (3n+1)\lfloor \log_2 n \rfloor - 3n + 6.$$

$$(3n+1)\lfloor \log_2 n \rfloor - 7n + 7 \leq$$
$$\leq C_{sort}(n) <$$
$$< (3n+1)\lfloor \log_2 n \rfloor - 3n + 6.$$

$$(3n+1)(\lfloor \log_2 n \rfloor - 2\frac{1}{3}) + 9\frac{1}{3} \leq$$
$$\leq C_{sort}(n) <$$
$$< (3n+1)(\lfloor \log_2 n \rfloor - 1) + 5.$$

One can show

$$(3n+1)\log_2 n - 7n + 5 < C_{sort}(n) <$$
$$< (3n+1)\log_2 n - 5.73n + 6.$$

1.8 The running time of MakeHeap

```
45      public void MakeHeap()
46      {
47          for (int i = N/2; i > 0; i--) FixHeap(i);
48      }
```

Here is a faster Heapsort.

```
66      public static void Heapsort(int[] array)
67      {
68          HeapSort H = new HeapSort(array); // uses array to store the eleme
69          H.MakeHeap(); // heap-construction phase
70          H.RemoveAll(); // heap-deconstruction phase
71      }
```

```
45          public void MakeHeap()
46          {
47              for (int i = N/2; i > 0; i--) FixHeap(i);
48          }
```

```
45          public void MakeHeap()
46          {
47              for (int i = N/2; i > 0; i--) FixHeap(i);
48          }
```

$$C_{\texttt{MakeHeap}}(n) = \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} C_{\texttt{FixHeap}(i)}(n)$$

$$C_{\texttt{MakeHeap}}(n) = \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} C_{\texttt{FixHeap}(i)}(n)$$

Recall

$$2 \times D_n - 1 \leq C_{\texttt{remove}}(n+1) \leq 2 \times D_n,$$

$$C_{\texttt{MakeHeap}}(n) = \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} C_{\texttt{FixHeap}(i)}(n)$$

$$2 \times D_n - 1 \leq C_{\texttt{remove}}(n+1) \leq 2 \times D_n,$$

$$\begin{gathered} C_{\texttt{FixHeap}(i)}(n) \leq \\ \leq C_{\texttt{remove}}(n+1) - 2 \times D_i \leq \\ \leq 2 \times (D_n - D_i), \end{gathered}$$

$$C_{\texttt{MakeHeap}}(n) = \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} C_{\texttt{FixHeap}(i)}(n)$$

$$C_{\texttt{FixHeap}(i)}(n) \leq 2(D_n - D_i)$$

$$C_{\texttt{MakeHeap}}(n) = \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} C_{\texttt{FixHeap}(i)}(n)$$

$$C_{\texttt{FixHeap}(i)}(n) \leq 2(D_n - D_i)$$

$$C_{\texttt{FixHeap}(i)}(n) \leq 2(\lfloor \log_2 n \rfloor - \lfloor \log_2 i \rfloor)$$

$$C_{\texttt{MakeHeap}}(n) = \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} C_{\texttt{FixHeap}(i)}(n)$$

$$C_{\texttt{FixHeap}(i)}(n) \leq 2(\lfloor \log_2 n \rfloor - \lfloor \log_2 i \rfloor)$$

$$C_{\texttt{MakeHeap}}(n) \leq \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} 2(\lfloor \log_2 n \rfloor - \lfloor \log_2 i \rfloor)$$

$$C_{\texttt{MakeHeap}}(n) \leq \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} 2(\lfloor \log_2 n \rfloor - \lfloor \log_2 i \rfloor)$$

$$C_{\texttt{MakeHeap}}(n) \leq 2(\lfloor \frac{n}{2} \rfloor \lfloor \log_2 n \rfloor - \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \lfloor \log_2 i \rfloor)$$

$$C_{\texttt{MakeHeap}}(n) \leq 2(\lfloor \frac{n}{2} \rfloor \lfloor \log_2 n \rfloor - \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \lfloor \log_2 i \rfloor)$$

$$\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \lfloor \log_2 i \rfloor =$$

$$C_{\texttt{MakeHeap}}(n) \leq 2(\lfloor\frac{n}{2}\rfloor\lfloor\log_2 n\rfloor - \sum_{i=1}^{\lfloor\frac{n}{2}\rfloor}\lfloor\log_2 i\rfloor)$$

$$\sum_{i=1}^{\lfloor\frac{n}{2}\rfloor}\lfloor\log_2 i\rfloor =$$

$$= \lfloor\frac{n}{2}\rfloor\lfloor\log_2 n\rfloor - \lfloor\frac{n}{2}\rfloor - 2^{\lfloor\log_2 n\rfloor} + \lfloor\log_2 n\rfloor + 1$$

$$C_{\texttt{MakeHeap}}(n) \leq 2(\lfloor\frac{n}{2}\rfloor\lfloor\log_2 n\rfloor - \sum_{i=1}^{\lfloor\frac{n}{2}\rfloor}\lfloor\log_2 i\rfloor)$$

$$\sum_{i=1}^{\lfloor\frac{n}{2}\rfloor}\lfloor\log_2 i\rfloor =$$

$$= \lfloor\frac{n}{2}\rfloor\lfloor\log_2 n\rfloor - \lfloor\frac{n}{2}\rfloor - 2^{\lfloor\log_2 n\rfloor} + \lfloor\log_2 n\rfloor + 1$$

Exercise: Prove it using the previous formula for $\Sigma_{i=1}^{n}\lfloor\log_2 i\rfloor$.

$$C_{\texttt{MakeHeap}}(n) \leq 2(\lfloor\frac{n}{2}\rfloor\lfloor\log_2 n\rfloor - \sum_{i=1}^{\lfloor\frac{n}{2}\rfloor}\lfloor\log_2 i\rfloor)$$

$$\sum_{i=1}^{\lfloor\frac{n}{2}\rfloor}\lfloor\log_2 i\rfloor =$$

$$= \lfloor\frac{n}{2}\rfloor\lfloor\log_2 n\rfloor - \lfloor\frac{n}{2}\rfloor - 2^{\lfloor\log_2 n\rfloor} + \lfloor\log_2 n\rfloor + 1$$

$$C_{\texttt{MakeHeap}}(n) \leq$$

$$2(\lfloor\frac{n}{2}\rfloor\lfloor\log_2 n\rfloor +$$

$$-(\lfloor\frac{n}{2}\rfloor\lfloor\log_2 n\rfloor - \lfloor\frac{n}{2}\rfloor - 2^{\lfloor\log_2 n\rfloor} + \lfloor\log_2 n\rfloor + 1))$$

$$C_{\texttt{MakeHeap}}(n) \leq$$
$$2(\lfloor\frac{n}{2}\rfloor\lfloor\log_2 n\rfloor+$$
$$-(\lfloor\frac{n}{2}\rfloor\lfloor\log_2 n\rfloor-\lfloor\frac{n}{2}\rfloor-2^{\lfloor\log_2 n\rfloor}+\lfloor\log_2 n\rfloor+1))$$

$$C_{\texttt{MakeHeap}}(n) \leq$$
$$2(\lfloor\frac{n}{2}\rfloor + 2^{\lfloor\log_2 n\rfloor} - \lfloor\log_2 n\rfloor - 1)$$

$$C_{\texttt{MakeHeap}}(n) \leq$$
$$2(\lfloor\frac{n}{2}\rfloor + 2^{\lfloor\log_2 n\rfloor} - \lfloor\log_2 n\rfloor - 1)$$

$$C_{\texttt{MakeHeap}}(n) \leq 2\frac{n}{2}+2\times 2^{\log_2 n}-2\log_2 n-2$$

$$C_{\texttt{MakeHeap}}(n) \leq 2\frac{n}{2}+2\times 2^{\log_2 n}-2\log_2 n-2$$

$$C_{\texttt{MakeHeap}}(n) \leq n + 2n - 2\log_2 n - 2$$

$$C_{\texttt{MakeHeap}}(n) \leq n + 2n - 2\log_2 n - 2$$

$$C_{\texttt{MakeHeap}}(n) \leq 3n - 2\log_2 n - 2$$

$$C_{\texttt{MakeHeap}}(n) \leq 3n - 2\log_2 n - 2$$

Note. The exact worst-case number of comparisons of `MakeHeap` is given by this formula:

$$C_{\texttt{MakeHeap}}(n) = 2n - 2s_2(n) - e_2(n),$$

where $s_2(n)$ is the sum of all digits of the binary representation of n and $e_2(n)$ is the exponent of 2 in the prime factorization of n.

It satisfies this inequality:

$$2n - 2\lg(n+1) \leq C_{\texttt{MakeHeap}}(n) \leq 2n - 4$$

for $n \geq 3$.

Below is a graph of $2n - 2s_2(n) - e_2(n)$ plotted below an upper bound $3n - 2\lg n + 2$.

1.9 The speed-up while using Make-Heap instead of H.insert() in a for-loop

The speed-up is equal to approximately 33%. Additional speed-up cuts the C_{remove} roughly by 50%, thus resulting in the total speed-up of *HeapSort* by about a factor of two. This brings the worst-case number of comparisons of key of *HeapSort* down to $n \log_2 n + o(n)$, that is, in par with *MergeSort*.

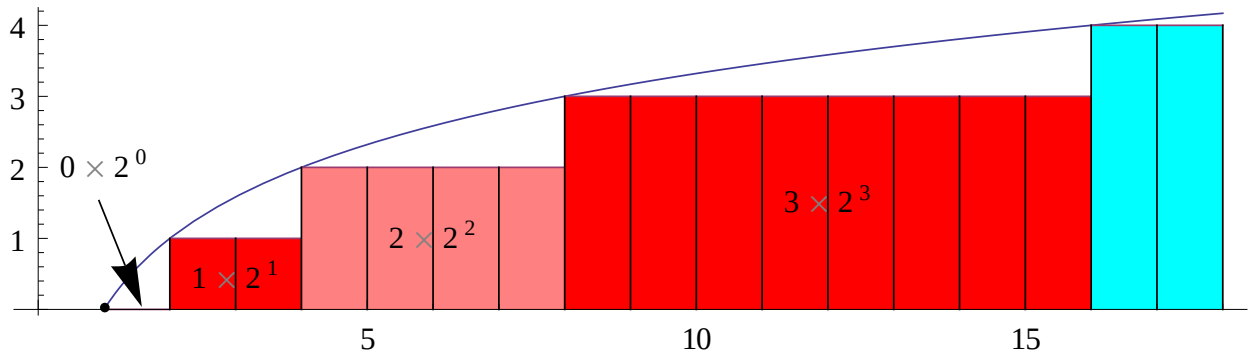


Figure 11: Computation of $\Sigma_{i=2}^{17}\lfloor\log_2 i\rfloor$ (the colored area) as $\Sigma_{i=1}^{\lfloor\log_2 17\rfloor-1} i \times 2^i$ (the reddish area) + $(17 - 2^{\lfloor\log_2 17\rfloor} + 1)\lfloor\log_2 17\rfloor$ (the cyan area).

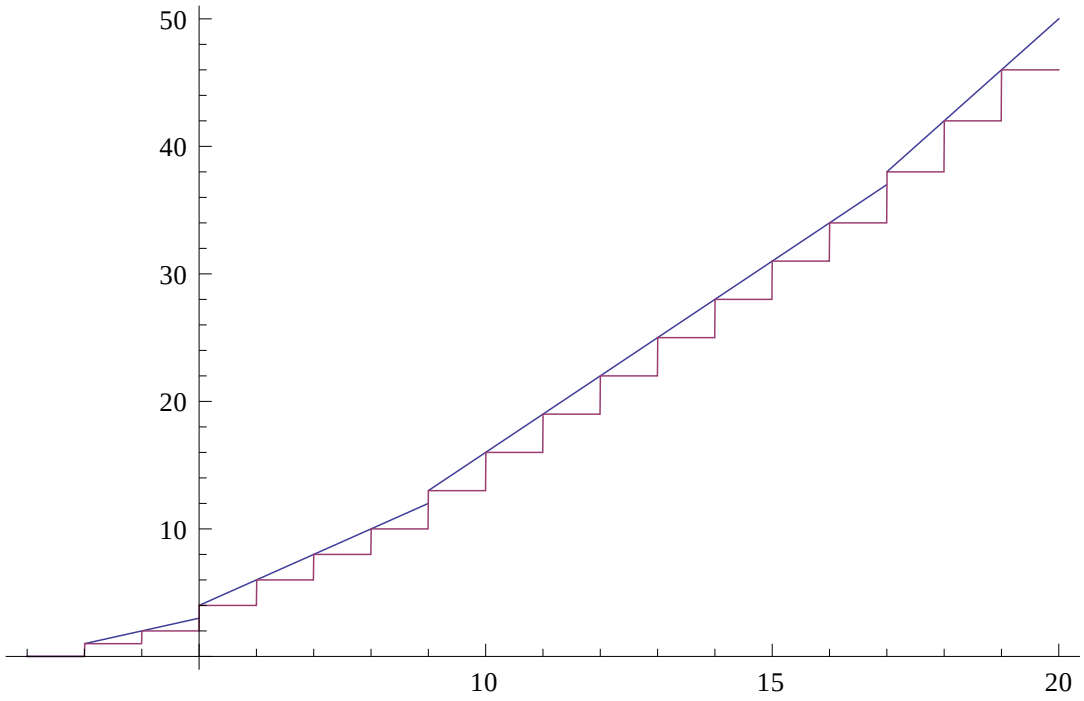


Figure 12: Functions $\sum_{i=2}^{n} \lfloor \lg i \rfloor$ and $(n+1)\lfloor \lg n \rfloor - 2^{\lfloor \lg n \rfloor + 1} + 2$.

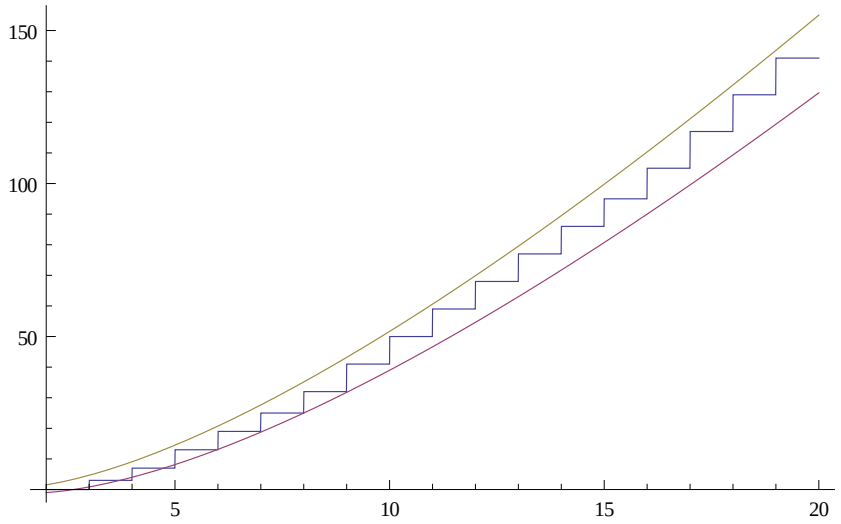


Figure 13: Graph of $C_{sort}(n)$ and its upper and lower bounds.

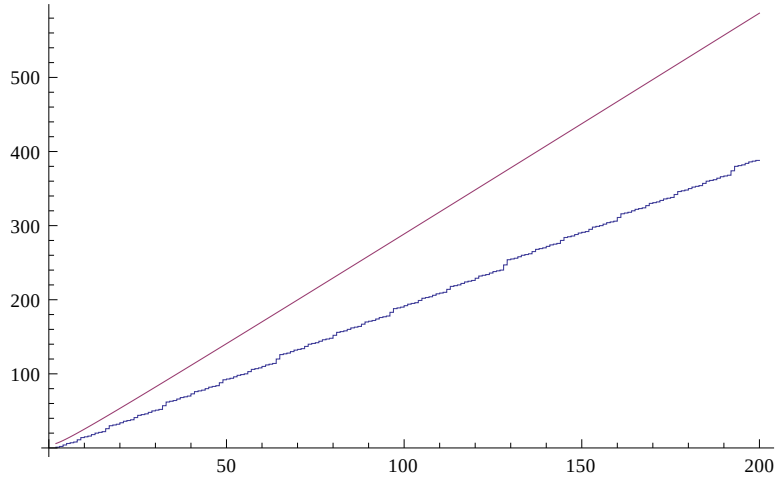


Figure 14: A graph of $2n - 2s_2(n) - e_2(n)$ plotted below an upper bound $3n - 2\lg n + 2$.