COMP20007 Design of Algorithms

Week 10 Workshop Solutions

Tutorial

1. Counting Sort To perform counting sort we note that the range of possible input characters are a, b, c, d, e, and f. We then loop through the array and tally the frequencies of each character:

```
  a b c d e f
[ 5 2 2 1 0 1 ]
```

We then read these frequencies in order to reconstruct a sorted array. That is, we take 5 `a`s, 2 `b`s *etc.*

```
[a a a a a b b c c d f]
```

2. Radix Sort Start by sorting by the final letter, keeping strings with the same final letter in the original relative order:

```
  abc bab cba ccc bbb aac abb bac bcc cab aba
cba aba | bab bbb abb cab | abc ccc aac bac bcc
```

We then join all of these together and sort by the middle letter:

```
  cba aba bab bbb abb cab abc ccc aac bac bcc
bab cab aac bac | cba aba bbb abb abc | ccc bcc
```

Finally by the first letter:

```
  bab cab aac bac cba aba bbb abb abc ccc bcc
aac aba abb abc | bab bac bbb bcc | cab cba ccc
```

So the final sorted array is

```
aac aba abb abc bab bac bbb bcc cab cba ccc
```

3. Stable Counting Sort To sort tuples by the first element and maintian relative order between tuples with equal first values we must ensure that *counting sort is stable.*

We can use the cumulative coints array based approach introduced in lectures to do this.

Alternatively, rather than storing frequencies in a frequency array, we have an array of linked lists with each object appended to the linked list corresponding to the value we're sorting by. The original array can be reconstructed by sequentially removing the head of the linked list and inserting into the new reconstructed array.

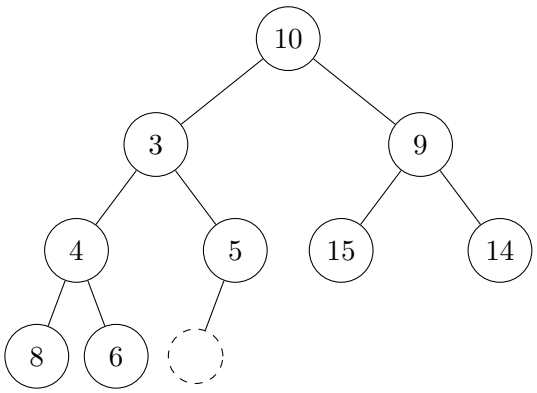
Both methods will require $\Theta(n + k)$ additional space, where k is the number of distinct values we're counting and n is the size of the input array.

4. Heaps

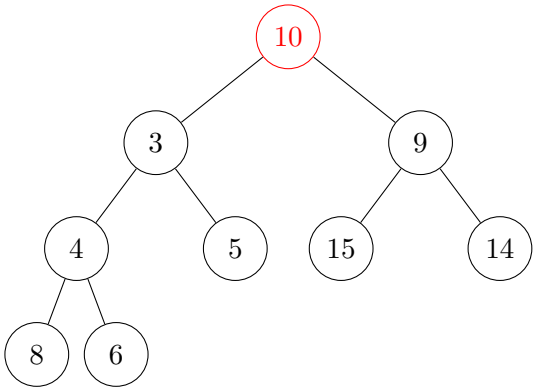
(a) As in the lectures we leave the first index empty, and then populate the array with the elements of the heap in level-order:

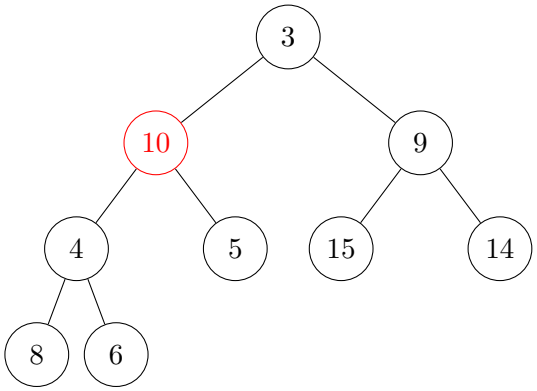
$$\Big[-, 1, 3, 9, 4, 5, 15, 14, 8, 6, 10\Big]$$

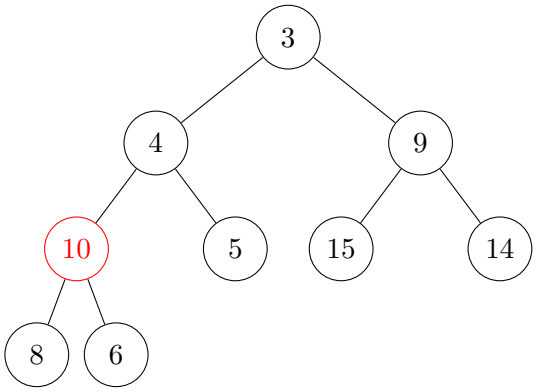
(b) First we store the value of the root of the heap, in this case it is 1, then we swap the last element in the heap to the root:

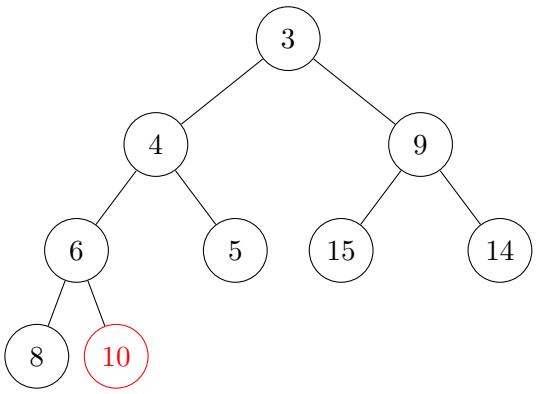


We then SiftDown the root, by comparing it to each of its children, and if it's larger than either swap it with the smaller child and repeat, if it's smaller than both children then stop.

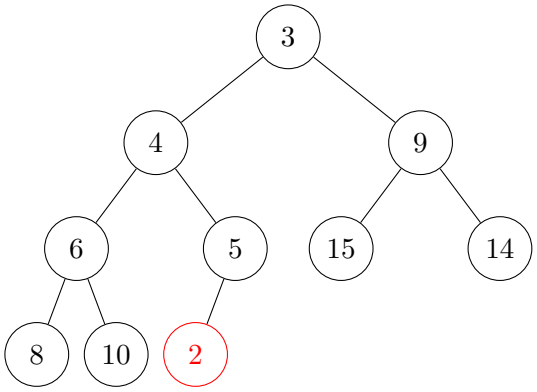


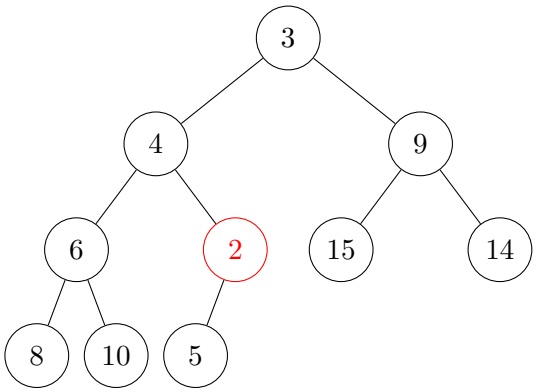


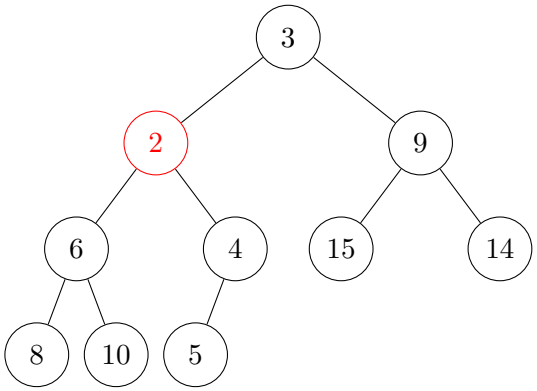


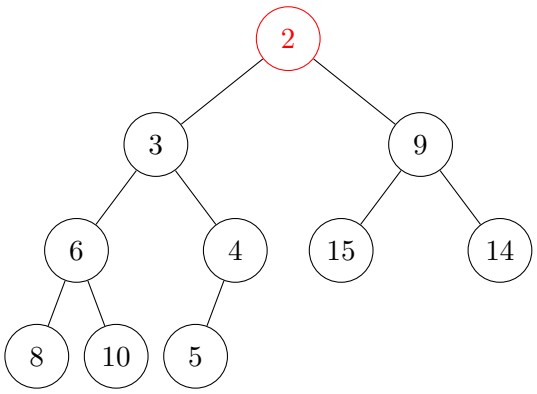


(c) We'll add the new value in a new node at the next spot in the tree, and then SiftUp. Sifting up consists of swapping the node with its parent as long its parent is larger than it.









5. (Optional) Using a min-heap for kth-smallest element Consider the following algorithm which finds the kth-smallest element using a min-heap.

function HEAPkTHSMALLEST($A[0 \dots n-1], k$)
  $Heap \leftarrow$ HEAPIFY($A[0 \dots n-1], k$)
  `// remove the smallest` $k-1$ `elements`
  for $i \leftarrow 0 \dots k-2$ **do**
    REMOVEMIN($Heap$)
  return REMOVEMIN($Heap$)

The time complexity of HEAPIFY is $\Theta(n)$ and each REMOVEMIN is $\Theta(\log n)$, so the total time complexity of this algorithm is $\Theta(n + k \log n)$.

Notice that this algorithm is not input-sensitive on the order of the array, it is only input sensitive with respect to k. This may be a desirable property for a variety of reasons.

6. (Revision) Recurrence Relations

(a) $T(1) = 1$
$T(n) = T(n/2) + 1$

$$\begin{aligned} T(n) &= T(n/2) + 1 \\ &= (T(n/4) + 1) + 1 \\ &= (T(n/8) + 1) + 1 + 1 \\ &\vdots \\ &= T(n/2^k) + k \\ &\vdots && \text{let } k = \log_2(n) \\ &= T(n/2^{\log_2(n)}) + \log_2(n) \\ &= T(n/n) + \log_2(n) \\ &= T(1) + \log_2(n) \\ &= 1 + \log_2(n) \end{aligned}$$

So $T(n) = 1 = \log_2(n) \in \Theta(\log n)$.

(b) $T(0) = 0$

$T(n) = T(n-1) + \frac{n}{5}$

$$
\begin{aligned}
T(n) &= T(n-1) + \frac{n}{5} \\
&= T(n-2) + \frac{n-1}{5} + \frac{n}{5} \\
&= T(n-3) + \frac{n-2}{5} + \frac{n-1}{5} + \frac{n}{5} \\
&\vdots \\
&= T(n-k) + \frac{n-(k-1)}{5} + \cdots + \frac{n-1}{5} + \frac{n}{5} \\
&\vdots && \text{let } k = n \\
&= T(n-n) + \frac{n-(n-1)}{5} + \cdots \frac{n-1}{5} + \frac{n}{5} \\
&= 0 + \frac{1}{5} + \cdots \frac{n-1}{5} + \frac{n}{5} \\
&= \frac{1}{5}\left(1 + \cdots + (n-1) + n\right) \\
&= \frac{n(n+1)}{10}
\end{aligned}
$$

So $T(n) = n(n+1)/10 \in \Theta(n^2)$.