*50.004 Introduction to Algorithms, Fall 2018*
Singapore University of Technology and Design 9 Oct, 2018
Gemma Roig, Ioannis Panageas, Subhajit Datta Home Exercises - Problem Set 1

---

# Student Information

Name: _____ Student ID: _____

Due Date: 01 Nov 11:59pm.

Submit answers on eDimension in pdf format. Submission without student information will **NOT** be marked! Any questions regarding the homework can be directed to the TA through email (contact information on eDimension).

---

# Exercise 1 Heap Sort

1. What are the minimum and maximum numbers of elements in a heap of height h?

2. What is the running time of HEAPSORT on an array A of length n that is already sorted in increasing order? What about decreasing order?

3. Where in a min-heap might the biggest element reside, assuming that all elements are distinct?

**Solution**:

1. Since a heap is an almost-complete binary tree (complete at all levels except possibly the lowest), it has at most $1 + 2 + 2^2 + 2^3 + \ + 2^h = 2^{h+1} - 1$ elements (if it is complete) and at least $2^h$ elements (if the lowest level has just 1 element and the other levels are complete).

2. The running time of HEAPSORT on an array of length that is already sorted in increasing order is $\Theta$(n log n), because even though it is already sorted, it will be transformed back into a heap and sorted.

   The running time of HEAPSORT on an array of length that is sorted in decreasing order will be $\Theta$((n log n). This occurs because even though the heap will be built in linear time, every time the element is removed and HEAPIFY is called, it could cover the full height of the tree.

3. The biggest element can only be one of leaf nodes. If not, it will have its own subtree and is smaller than any element on that subtree, which contradicts the fact that it is the biggest element.

# Exercise 2 Binary Search Tree

Suppose we have int values between 1 and 1000 in a BST and search for 525. Which of the following **cannot** be the sequence of keys examined, and **why**?

(a) 4 300 800 500 823 525

(b) 300 325 700 699 650 510 520 525

(c) 900 873 850 300 412 600 570 550 400 525

(d) 700 670 600 300 350 379 400 570 550 510 525

**Solution**: Draw the BST as if you were tracing the keys in the order given. (a) and (c) cannot be the sequence of keys examined. For sequence (a), element 823 cannot appear after 500. For sequence (c), element 400 cannot appear after 600, 570 and 550. Look at the figures below.

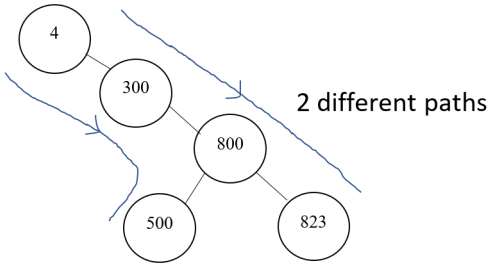


**Figure 1**: BST ques a

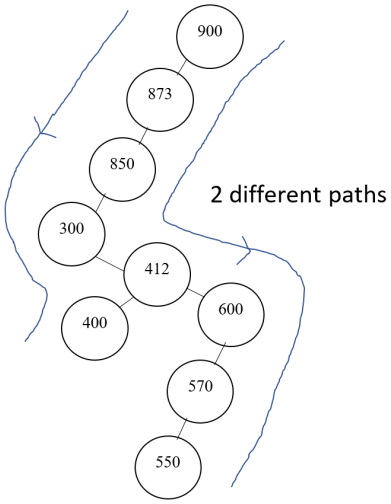


**Figure 2**: BST ques c

# Exercise 3 AVL Tree

1. Insert the following sequence of elements into an AVL tree (not ordinary BST), starting with an empty tree: 22; 33; 25; 38; 43; 28; 29; 31. Show steps by drawing.
2. Delete 28 from the AVL tree in question1. Show steps by drawing.
3. Delete 43 from the AVL in question1 (note that node 28 is added back to the AVL tree). Show steps by drawing.

**Solution**:

1. Refer to Figure 3.

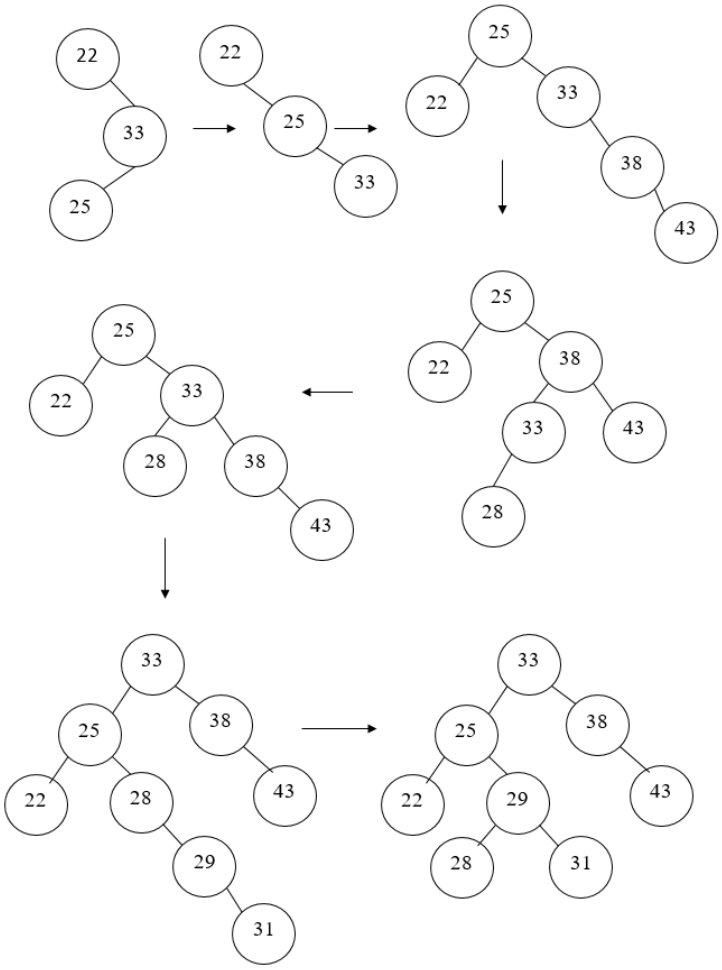


**Figure 3**: AVL ques 1

2. Node 28 is removed from the AVL tree. No other changes to the tree structure.
3. Refer to Figure 4.

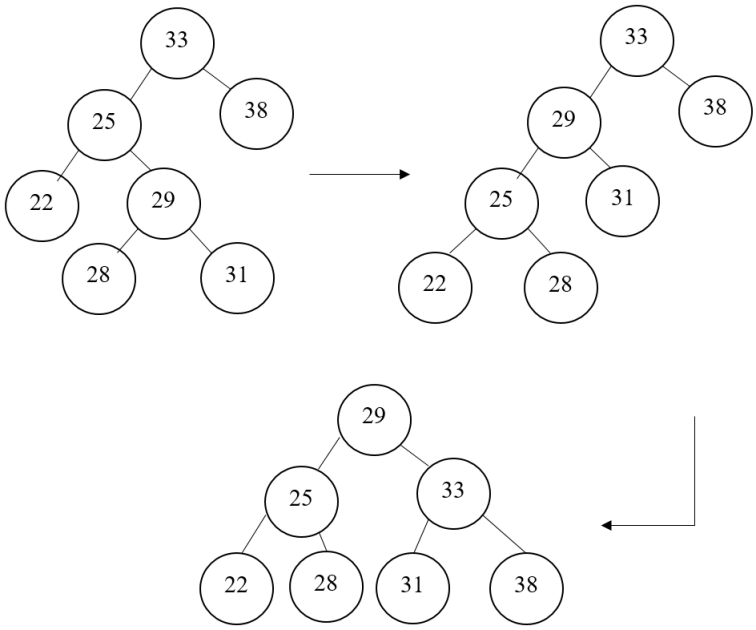


**Figure 4**: AVL ques3

# Exercise 4 Sorting

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

**Figure 5**: Counting sort (taken from CLRS book).

1. Refer to Figure 5 for the counting sort algorithm. Suppose we are asked to rewrite the **for loop** header in **line 10** of the counting sort algorithm as:

   ***for** j=1 **to** A.length*

Does the algorithm still works properly after this modification? Provide reasons.

2. Describe an algorithm that, given n integers in the range of 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range [a...b] in $O(1)$ time. Your algorithm should use $\Theta(n + k)$ preprocessing time. *Hint: use the counting-sort algorithm*.

3. Here are 11 different words: *hat, ten, hen, two, pan, one, tea, rat, rag, box, bat*. Sort the words in increasing alphabetical order with **radix sort**. Show all steps needed.

**Solution**:

1. The algorithm still works correctly. The order that elements are taken out of C and put into B does not affect the placement of elements with the same key. It will still fill the interval (C[k-1], C[k]] with elements of key k.

2. The algorithm will begin by preprocessing exactly as COUNTING-SORT does in lines 1 through 9, so that C[i] contains the number of elements less than or equal to i in the array. When queried about how many integers fall into a range [a..b], simply compute C[b]- C[a-1]. This takes O(1) times and yields the desired output.

3. hat tea rag bat
   ten one pan box
   hen rag hat hat
   two ten rat hen
   pan hen bat one
   one pan tea pan
   tea two ten rag
   rat hat hen rat
   rag rat one tea
   box bat box ten
   bat box two two

# Exercise 5 Hashing

Suppose that we are given a key $k$ to search for in a hash table with positions $0, 1, ..., m - 1$, and suppose that we have a hash function $h$ mapping the key space into the set $\{0, 1, ..., m - 1\}$. The search scheme is as follows:

1. Compute the value $j = h(k)$, and set $i = 0$.

2. Probe in position $j$ for the desired key $k$. If you find it, or if this position is empty, terminate the search.

3. Set $i = i + 1$. If $i$ now equals $m$, the table is full, so terminate the search. Otherwise, set $j = (i + j) \bmod m$, and return to step 2.

Assume that m is a power of 2.
Show that this scheme is an instance of the following general quadratic probing scheme,

$$h'(k, i) = (h(k) + c_1 i + c_2 i^2) \bmod m.$$

Find the constants $c_1$ and $c_2$.

**Solution**: From how the probe-sequence computation is specified, it is easy to see that the probe sequence is $h(k), h(k) + 1, h(k) + 1 + 2, h(k) + 1 + 2 + 3, ..., h(k) + 1 + 2 + 3 + ... + i, ...$, where all the arithmetic is modulo $m$. Starting the probe numbers from $0$, the $i$th probe is offset (modulo $m$) from $h(k)$ by

$$\sum_{j=0}^{i} j = \frac{i(i+1)}{2} = \frac{1}{2}i^2 + \frac{1}{2}i.$$

Thus, we can write the probe sequence as a quadratic probing of

$$h'(k, i) = (h(k) + \frac{1}{2}i + \frac{1}{2}i^2) \bmod m.$$

Therefore, $c_1 = \frac{1}{2}$ and $c_2 = \frac{1}{2}$.