EL-GY 9343 Homework 4

Yihao Wang

yw7486@nyu.edu

1. Demonstrate the operation of HOARE-PARTITION on the array $A = \{14, 12, 14, 19, 5, 3, 4, 14, 7, 22, 16\}$. Show the array after each iteration of the while loop in the lines of 4 to 11 in the code of lecture notes.

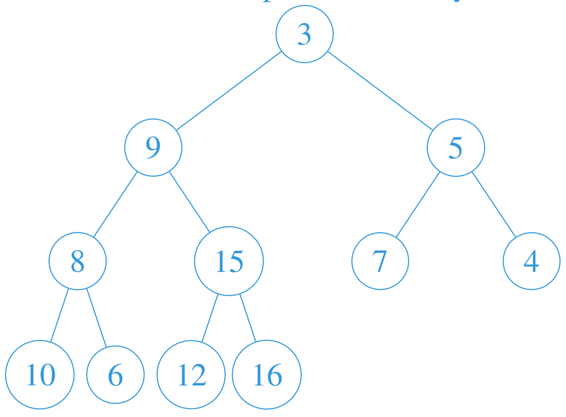
We choose the first element 14 as the pivot, and we use two underlines to denote the position of two pointers just before the *swap* operation. Then we have

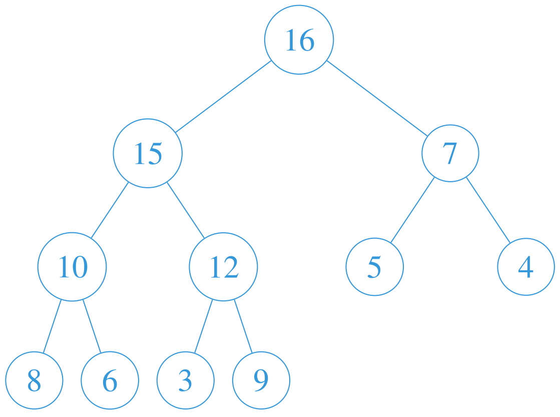
After i iterations	Array A
0	$\{\underline{14}, 12, 14, 19, 5, 3, 4, 14, \underline{7}, 22, 16\}$
1	$\{7, 12, \underline{14}, 19, 5, 3, 4, \underline{14}, 14, 22, 16\}$
2	$\{7, 12, 14, \underline{19}, 5, 3, \underline{4}, 14, 14, 22, 16\}$
3	$\{7, 12, 14, 4, 5, 3, \underline{\underline{19}}, 14, 14, 22, 16\}$

2. For the following array: $A = \{3, 9, 5, 8, 15, 7, 4, 10, 6, 12, 16\}$,

(a) Create a max heap using the algorithm BUILD-MAX-HEAP.

We first create a heap from the array A as left below:

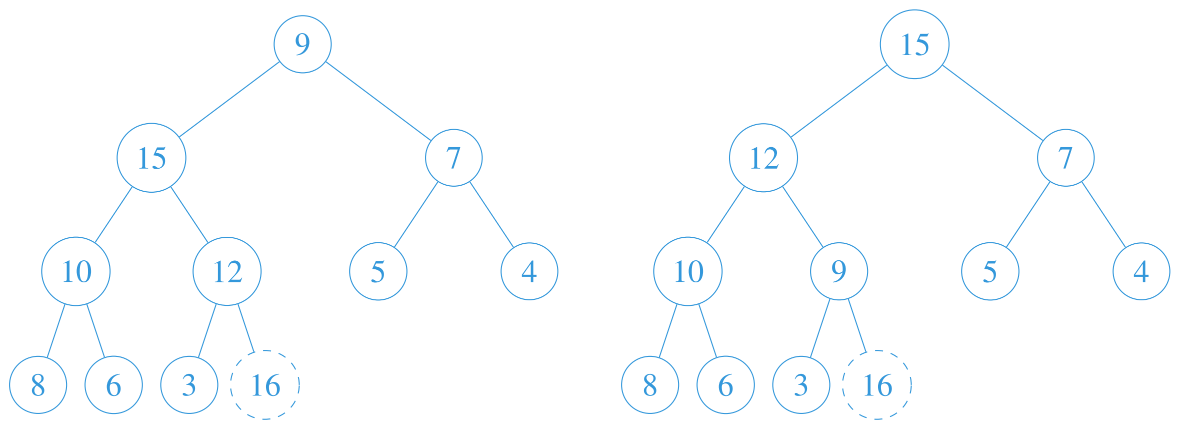




After calling MAX-HEAPIFY on the first half elements, i.e., {15, 8, 5, 9, 3} one after another, the heap becomes upper right, which is {16, 15, 7, 10, 12, 5, 4, 8, 6, 3, 9} in the array form.

(b) Remove the largest item from the max heap you created in 2(a), using the HEAP-EXTRACT-MAX function. Show the array after you have removed the largest item.

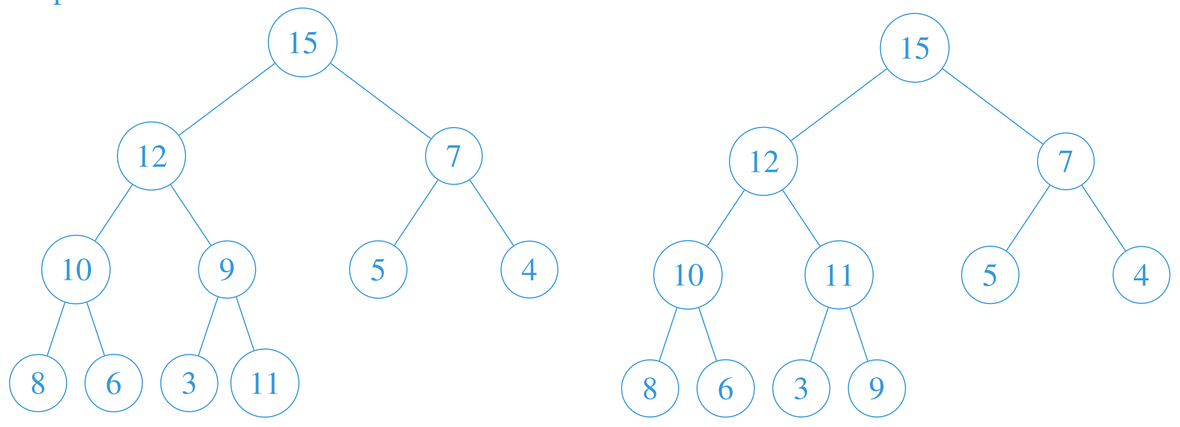
After exchange the last element with the max one and extract the max value, the heap becomes left below:



Then we call MAX-HEAPIFY on the new root, which turns the heap to upper right. In the array form, the heap is $\{15, 12, 7, 10, 9, 5, 4, 8, 6, 3\}$.

(c) Using the algorithm MAX-HEAP-INSERT, insert 11 into the heap that resulted from question 2(b). Show the array after insertion.

After append the array with $-\infty$, we call HEAP-INCERASE-KEY on the new element to increase its key to 11. Before recurrently exchange the last element with its parent, the heap is like left below



After the HEAP-INCERASE-KEY operation finishes, the heap is $\{15, 12, 7, 10, 11, 5, 4, 8, 6, 3, 9\}$ in the array form.

3. For an disordered array with n elements, design an algorithm for finding the median of this array. Your algorithm should traverse the array only once.

Notes: You can imagine the array as a flow which means you can get the data one by one, and you need to do some *cheap* operation at the time you see each element. The size of this array, n, is big and you know n from the start. Please do not sort the array, or you cannot get full mark. A hint to solve this problem is to use heap.

The median is basically the minimum one of the largest $\lfloor \frac{n}{2} \rfloor$ elements. So I propose Algo.1, whose key idea is to maintain one MaxHeap and one MinHeap at the same time.

Algorithm 1 Find Median Using Heap(A)

Input: A as input array to find the median, containing n real numbers

Output: The median of A

```
1:  $B$ : MaxHeap $\leftarrow \{\}$, $C$ : MinHeap $\leftarrow \{\}$
2:  for $i = 1, 2, \ldots, n$ do
3:      MaxHeapInsert$(B, A[i])$
4:      if $i > \lfloor \frac{n}{2} \rfloor$ then
5:          $m \leftarrow$ HeapExtractMax$(B)$          ▷ $B$ keeps a size of $\lfloor \frac{n}{2} \rfloor$, thus efficient
6:          MinHeapInsert$(C, m)$
7:      end if
8:  end for
9:  $m' \leftarrow$ HeapExtractMin$(C)$
10: if $n$ mod $2 == 0$ then
11:     $m \leftarrow$ HeapExtractMax$(B)$
12:     return $(m' + m)/2$
13: else
14:     return $m'$
15: end if
```

4. Finding the median of an unordered array in $O(n)$ (Part II). In last homework, we looked at an algorithm that tried to find the median in $O(n)$, but it is not correct. This time we are going to fix it.

Let's consider a more general problem: given an unsorted array L of n elements ($L[1, ..., n]$), how to find the k^{th} smallest element in it (and when $k = \lceil \frac{n}{2} \rceil$, this turns out to find the median). We can also do divide-and-conquer to solve it, by the algorithm called QUICKSELECT, as follows.

```
QUICKSELECT$(L, p, r, k)$
$q \leftarrow$ PARTITION$(L, p, r)$
$t \leftarrow q - p + 1$
if k == t then
    return $L[q]$
else if $k < t$ then
    QUICKSELECT$(L, p, q-1, k)$ {Only look at the left part}
else
    QUICKSELECT$(L, q+1, r, k-t)$ {Only look at the right part}
end if
```

(a) The pivot selection in PARTITION plays a key role in optimizing the performance of teh QUICKSORT algorithm. If we use HOARE-PARTITION as the PARTITION function, please **solve** for the worst-case running time. And, what is the average running time (just give the answer)?

The worst-case occurs when the array is already in order, which results in a maximally unbalanced partition for every single partition. Therefore, the recursion formula is

$$T(n) = \Theta(1) + T(n-1) + \Theta(n) = T(n-1) + \Theta(n)$$

We can find the k^{th} smallest element after k levels of recursion. Therefore, $C(n,k)$, the total running time of finding the k^{th} smallest element in an array of n, is

$$\begin{aligned} C(n,k) &\simeq n + (n-1) + (n-2) + \cdots + (n-k) \\ &= \sum_{i=1}^{n} i - \sum_{j=1}^{n-k-1} j = \frac{n(n+1)}{2} - \frac{(n-k-1)(n-k)}{2} \\ &= \underline{-\frac{1}{2}k^2 + (n-\frac{1}{2})k + n} \end{aligned}$$

The average running time can be calculated by taking expectation of $C(n,k)$ on k:

$$\begin{aligned} \mathbb{E}_k[C(n,k)] &= \mathbb{E}_k[-\frac{1}{2}k^2 + (n-\frac{1}{2})k + n] \\ &= -\frac{1}{2}\mathbb{E}[k^2] + (n-\frac{1}{2})\mathbb{E}[k] + n \\ &= -\frac{1}{2}\frac{(n+1)(2n+1)}{6} + (n-\frac{1}{2})\frac{n}{2} + n \sim \underline{\Theta(n^2)} \end{aligned}$$

(b) The b^* found in the algorithm in last homework may not be the true median, yet it could serve as a good pivot. Let's look at the BFPRT algorithm (a.k.a. median-of-medians), which uses this pivot to do the partition, as follows.

```
**BFPRT**$(L, p, r)$
Divide $L[p, ..., r]$ into $\frac{r-p+1}{5}$ lists of size 5 each
Sort each list, let $i^{th}$ list be $a_i \le a'_i \le b_i \le c_i \le c'_i, i = 1, 2, \dots, \frac{r-p+1}{5}$    $\triangleright \Theta(\frac{n}{5})$
Recursively find median of $b_1, b_2, ..., b_{\frac{r-p+1}{5}}$, call it $b^*$    $\triangleright T(\frac{n}{5})$
Use $b^*$ as the pivot and reorder the list (do swapping like in HOARE-PARTITION after pivot selection)    $\triangleright \Theta(n)$
Suppose after reordering, $b^*$ is at $L[q]$, **return** q
```

Solve for the worst-time running time of the QUICKSELECT algorithm, if we utilize BFPRT as PARTITION.

(**Hints**: In QUICKSORT, the worst-time running time occurs when every partition is extremely unbalanced, so does in QUICKSELECT. Therefore, we need to consider how unbalanced this partition could be. Remember that there is a median finding step in BFPRT algorithm. And in this question we only require the solution in big-O notation.)

According to the selection of pivot b^* in the BFPRT algorithm, we know that $b^* \geq \{b_1, b_2, \ldots, b_{\frac{r-p+1}{10}}\}$ which are $\frac{1}{10}n$ numbers. Moreover, $b_i \geq a_i, a_i^{'}$ for all $i = 1, 2, \ldots, \frac{r-p+1}{10}$, which are $2 * \frac{r-p+1}{10} = \frac{2}{10}n$ numbers. Therefore, b^* is greater or equals to at least $\frac{3}{10}n$ numbers, which means there are still $\frac{7}{10}n$ numbers left. Now we have the recursion equation in the worst case:

$$T(n) = \underbrace{T(\frac{1}{5}n)}_{\text{Recursively finding median}} + \overbrace{T(\frac{7}{10}n)}^{\text{Worst-case partition}} + \underbrace{O(n)}_{\text{Sorting and reordering list}}$$

By solving this recursion, we can tell that $\underline{T(n) \sim O(n)}$ in the worst case because the sum of all coefficient is $\frac{1}{5} + \frac{7}{10} = \frac{9}{10} < 1$.