EL9343 Homework 4

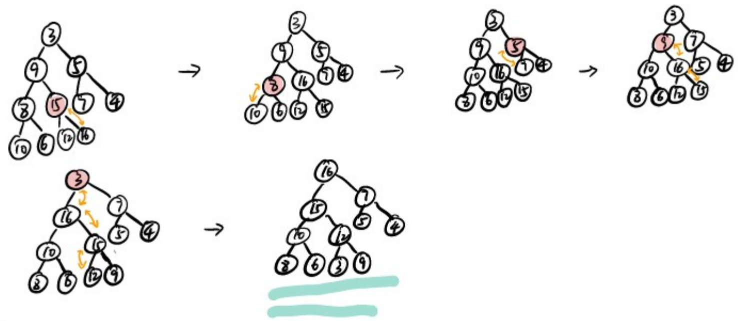
Due: Oct. 5th 8:00 a.m.

1. Demonstrate the operation of HOARE-PARTITION on the array $A =< 14, 12, 19, 5, 6, 9, 3, 4, 13, 7, 22, 16 >$. Show the array after each iteration of the while loop in the lines of 4 to 11 in the code of lecture notes.
   Solution:
   $x = A[1] = 14$
   $i = 0, j = 12$
   The first iteration:
   $j = 10$: $A[10] = 7 < x$
   $i = 1$: $A[1] = 14 \geq x$
   After exchanging, $A =< 7, 12, 19, 5, 6, 9, 3, 4, 13, 14, 22, 16 >$
   The second iteration:
   $j = 9$: $A[9] = 13 < x$
   $i = 3$: $A[3] = 19 > x$
   After exchanging, $A =< 7, 12, 13, 5, 6, 9, 3, 4, 19, 14, 22, 16 >$
   The third iteration:
   $j = 8i$: $A[8] = 4 < x$
   $i = 9$: $A[9] = 19 > x$
   Because $i > j$, no exchange. $A =< 7, 12, 13, 5, 6, 9, 3, 4, 19, 14, 22, 16 >$
   Loop ends.

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

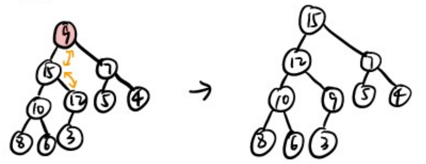
   Solution:

(a)

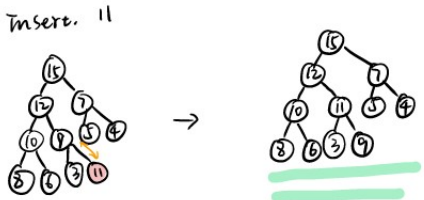


(b)

Max - extract



Insert. 11



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. 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.
Solution:
Other reasonable solutions can also get full marks. Description is enough. Pseudo-code is not necessary.

FIND-MEDIAN-1(A)
Build a MIN-HEAP using first $\lfloor \frac{n}{2} \rfloor + 1$ elements of A
for element e in the other half of A **do**
  if $e >$ MIN-HEAP[1] (MIN-HEAP[1] is the root value of the heap) **then**
    MIN-HEAP[1] $= e$, then do MIN-HEAPIFY
  end if
end for
if n%2==0 **then**
  return (MIN-HEAP[1]+MIN-HEAP[2])/2 (root and second values of the heap)
else
  return MIN-HEAP[1] (root value of the heap)
end if

Also we can use a MIN-HEAP and a MAX-HEAP, starting from empty. Go through the array, add each element e to (i) MIN-HEAP if $e > MIN-HEAP[1]$; (ii) MAX-HEAP if $e \leq MAX-HEAP[1]$. At each step, keep the two heaps balanced: if unbalanced, extract the root value of the larger heap and add it to the smaller heap. After scanning the array, return the root value of the larger heap (if n is odd) or the average of two heap roots (if n is even).

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 ($L[1, \ldots, n]$), 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) Since we have learned about the QUICKSORT algorithm, we should understand that the pivot selection in PARTITION plays a key role in optimizing the performance. If we use HOARE-PARTITION as the PARTITION function, please solve for the worst-case running time.
Solution:
In worst-case, every partition is extremely unbalanced. Let $T(n)$ be the running time of the algorithm for a list of n elements, then in the worst-case,

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

By iteration method, we can show that $T(n) = \Theta(n^2)$, in worst-case.

(b) Please guess on the average running time of the algorithm (still using HOARE-PARTITION, just guess about the answer and no need to verify it).
Solution:
In average the algorithm runs in $O(n)$ time. (Just like QUICKSORT, fast in average, but not promising in worst-case.)

(c) 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 $L[p, \dots, r]$ into $\frac{r-p+1}{5}$ lists of size 5 each
Sort each list, let i^{th} list be $a_i \le a'_i \le b_i \le c_i \le c'_i, i = 1, 2, \dots, \frac{r-p+1}{5}$
Recursively find median of $b_1, b_2, \dots, b_{\frac{r-p+1}{5}}$, call it b^*
Use b^* as the pivot and reorder the list (do swapping like in HOARE-PARTITION after pivot selection)
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.)

Solution:
Remember there will always be $\frac{3}{10}n$ numbers larger than b^* and some other $\frac{3}{10}n$ numbers smaller than b^*. Thus the most unbalanced case for the BFPRT algorithm is always returning the $7:3$ partition, and the target k-th smallest element always in $\frac{7}{10}$ part. There is a median finding of $\frac{n}{5}$ elements inside BFPRT. The recursive we have is,

$$T(n) = T(\frac{7}{10}n) + T(\frac{n}{5}) + O(n)$$

By substitution method, we could prove that $T(n) = O(n)$, which means the algorithm runs in $O(n)$, even in worst-case.

Special Notes: This algorithm is interesting theoretically, but is not commonly used in practice. Running of it could cause at most $22n$ comparisons and $\frac{37}{2}n$ swaps.
There are researchers trying to improve the performance. One way is to use randomized pivot normally, and only to switch to b^* pivot when "necessary". Others improvements are on how to set groups. Please refer to the blog[1] and the paper[2] (mentioned in the blog), if you are interested in such topics.

[1] https://rcoh.me/posts/linear-time-median-finding/

[2] http://erdani.org/research/sea2017.pdf