Fundamental Algorithms - Spring 2018
Homework 2

Daniel Rivera Ruiz
Department of Computer Science
New York University
drr342@nyu.edu

1. Using the method explained in class:
   First we create an auxiliary array B of size $r - p + 1 = 12$.
   Then we select the pivot as the r^{th} element of A and initialize *left* and *right*:

$$\text{pivot} = x = A[12] = 10$$
$$\text{left} = p = 1$$
$$\text{right} = r = 12$$

   For each element $A[i]$ of A, with i ranging between $p = 1$ and $r - 1 = 11$, we traverse A, compare the values with the pivot and update the values of B, *left* and *right*:

i	$A[i]$	$\leq x$?	B	left	right
1	13	False	(_, _, _, _, _, _, _, _, _, _, 13)	1	11
2	18	False	(_, _, _, _, _, _, _, _, _, _, 18, 13)	1	10
3	9	True	(9, _, _, _, _, _, _, _, _, _, 18, 13)	2	10
4	5	True	(9, 5, _, _, _, _, _, _, _, _, 18, 13)	3	10
5	12	False	(9, 5, _, _, _, _, _, _, _, 12, 18, 13)	3	9
6	8	True	(9, 5, 8, _, _, _, _, _, _, 12, 18, 13)	4	9
7	7	True	(9, 5, 8, 7, _, _, _, _, _, 12, 18, 13)	5	9
8	4	True	(9, 5, 8, 7, 4, _, _, _, _, 12, 18, 13)	6	9
9	11	False	(9, 5, 8, 7, 4, _, _, _, 11, 12, 18, 13)	6	8
10	2	True	(9, 5, 8, 7, 4, 2, _, _, 11, 12, 18, 13)	7	8
11	6	True	(9, 5, 8, 7, 4, 2, 6, _, 11, 12, 18, 13)	8	8

   We copy the pivot value in the last position available in B, which will be $i = \textit{left}$:

$$B[8] = 10$$

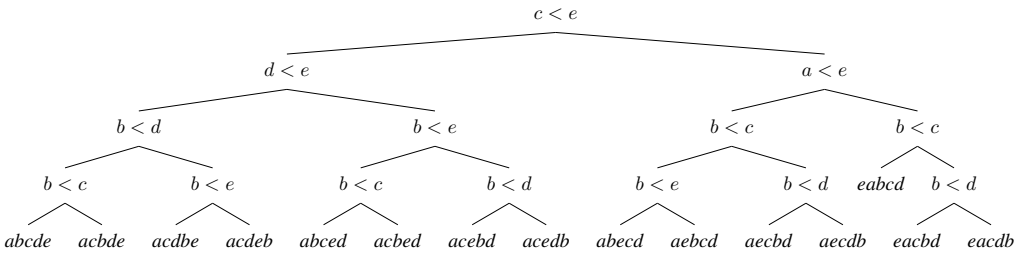
   Finally, we copy back the partitioned array B into A and return $\textit{left} = 8$.

2. Let's use the following table to calculate the exact value of $L(1023)$:

Iteration	No. of pivots	No. of sub-arrays	Size of sub-array	No. of comparisons
1	1	1	1022	$1 \cdot 1022 = 1022$
2	3	2	510	$2 \cdot 510 = 1020$
3	7	4	254	$4 \cdot 254 = 1016$
4	15	8	126	$8 \cdot 126 = 1008$
5	31	16	62	$16 \cdot 62 = 992$
6	63	32	30	$32 \cdot 30 = 960$
7	127	64	14	$64 \cdot 14 = 896$
8	255	128	6	$128 \cdot 6 = 768$
9	511	256	2	$256 \cdot 2 = 512$
			Total	**8194**

The 10^{th} (last) iteration will not require any comparisons, because *all* the values in the array will be pivots and therefore will not be compared to anything. Finally, we have the desired value $L(1023) = 8194$.

3. The following decision tree solves the problem with (at most) four further comparisons (the left branch means that the comparison yields true):



4. If $a < c =$ *false*, then Babu already knows that $c < a < b$ after using two comparisons. The third comparison could either be "is $c < d$?" or "is $c < e$?", and he will end up in a situation similar to the one depicted in problem 3. With four questions left, Babu could assure to sort all the elements. Let us then focus on the case where $a < c =$ *true*:
   (a) At this point, after two comparisons, Babu knows that $a < c$ and $a < b$.
   (b) The next "smart" question would be "is $c < d$?". If the answer to this question were *true*, after three comparisons Babu would know that $(a < c < d)$ and $(a < b)$, and he would be able to follow the procedure from problem 3. (The question could also be "is $c < e$?" with similar results).
   (c) If $c < d =$ *false*, after three questions Babu only knows that $a < c$, $a < b$ and $d < c$. At this point, he will need one extra question to be able to get a triple inequality (relating either abc or acd) that will leave him in a similar position to the one of problem 3.
   (d) With four questions already used and only three questions left, Babu will not be able to guarantee the sorting of all the elements (although he might get lucky and follow the one path in the tree that only requires three comparisons).

5. First we create the auxiliary array C of size $k+1$ with $k=6$ and fill it with zeros:

$$C = (0, 0, 0, 0, 0, 0, 0)$$

Then we populate C by traversing A from $s=1$ to $s=N=11$:

s	$A[s]$	C
1	6	(0, 0, 0, 0, 0, 0, 1)
2	0	(1, 0, 0, 0, 0, 0, 1)
3	2	(1, 0, 1, 0, 0, 0, 1)
4	2	(1, 0, 2, 0, 0, 0, 1)
5	0	(2, 0, 2, 0, 0, 0, 1)
6	1	(2, 1, 2, 0, 0, 0, 1)
7	3	(2, 1, 2, 1, 0, 0, 1)
8	4	(2, 1, 2, 1, 1, 0, 1)
9	6	(2, 1, 2, 1, 1, 0, 2)
10	1	(2, 2, 2, 1, 1, 0, 2)
11	3	(2, 2, 2, 2, 1, 0, 2)

Then we create the cumulative version of C:

t	C
1	(2, 4, 2, 2, 1, 0, 2)
2	(2, 4, 6, 2, 1, 0, 2)
3	(2, 4, 6, 8, 1, 0, 2)
4	(2, 4, 6, 8, 9, 0, 2)
5	(2, 4, 6, 8, 9, 9, 2)
6	(2, 4, 6, 8, 9, 9, 11)

Finally, we populate the auxiliary array B:

j	$A[j]$	$C[A[j]]$	B	C (updated)
11	3	8	(_, _, _, _, _, _, _, 3, _, _, _)	(2, 4, 6, 7, 9, 9, 11)
10	1	4	(_, _, _, 1, _, _, _, 3, _, _, _)	(2, 3, 6, 7, 9, 9, 11)
9	6	11	(_, _, _, 1, _, _, _, 3, _, _, 6)	(2, 3, 6, 7, 9, 9, 10)
8	4	9	(_, _, _, 1, _, _, _, 3, 4, _, 6)	(2, 3, 6, 7, 8, 9, 10)
7	3	7	(_, _, _, 1, _, _, 3, 3, 4, _, 6)	(2, 3, 6, 6, 8, 9, 10)
6	1	3	(_, _, 1, 1, _, _, 3, 3, 4, _, 6)	(2, 2, 6, 6, 8, 9, 10)
5	0	2	(_, 0, 1, 1, _, _, 3, 3, 4, _, 6)	(1, 2, 6, 6, 8, 9, 10)
4	2	6	(_, 0, 1, 1, _, 2, 3, 3, 4, _, 6)	(1, 2, 5, 6, 8, 9, 10)
3	2	5	(_, 0, 1, 1, 2, 2, 3, 3, 4, _, 6)	(1, 2, 4, 6, 8, 9, 10)
2	0	1	(0, 0, 1, 1, 2, 2, 3, 3, 4, _, 6)	(0, 2, 4, 6, 8, 9, 10)
1	6	10	(0, 0, 1, 1, 2, 2, 3, 3, 4, 6, 6)	(0, 2, 4, 6, 8, 9, 9)

And we copy back the sorted array B into A.

6. If we retrieve the third largest entry of the Max-Heap by performing `EXTRACT-MAX` twice and `MAX` once, the time complexity will be:

$$T = O(lg(N)) + O(lg(N)) + O(1)$$
$$\boldsymbol{T = O(lg(N))}$$

However, if we look only at the top seven elements of the heap, we can guarantee that the third largest entry will be among them. We can then sort this elements (which will take constant time) and return the 5^{th} element of the sorted array, i.e., the third largest element of the original Max-Heap.

Given the original Max-Heap in the form of an array $A[1...N]$ the algorithm is as follows:

(i) Copy $A[i]$ for $i = 1$ to $i = 7$ into an auxiliary array B. (*Time* $= O(1)$)

(ii) Sort the array B, which only has 7 elements. (*Time* $= O(1)$)

(iii) Return the 5^{th} element of B (*Time* $= O(1)$)

Since all of the steps in this algorithm require a constant amount of time, the overall complexity of the algorithm is $O(1)$, which is clearly better than $O(lg(N))$.