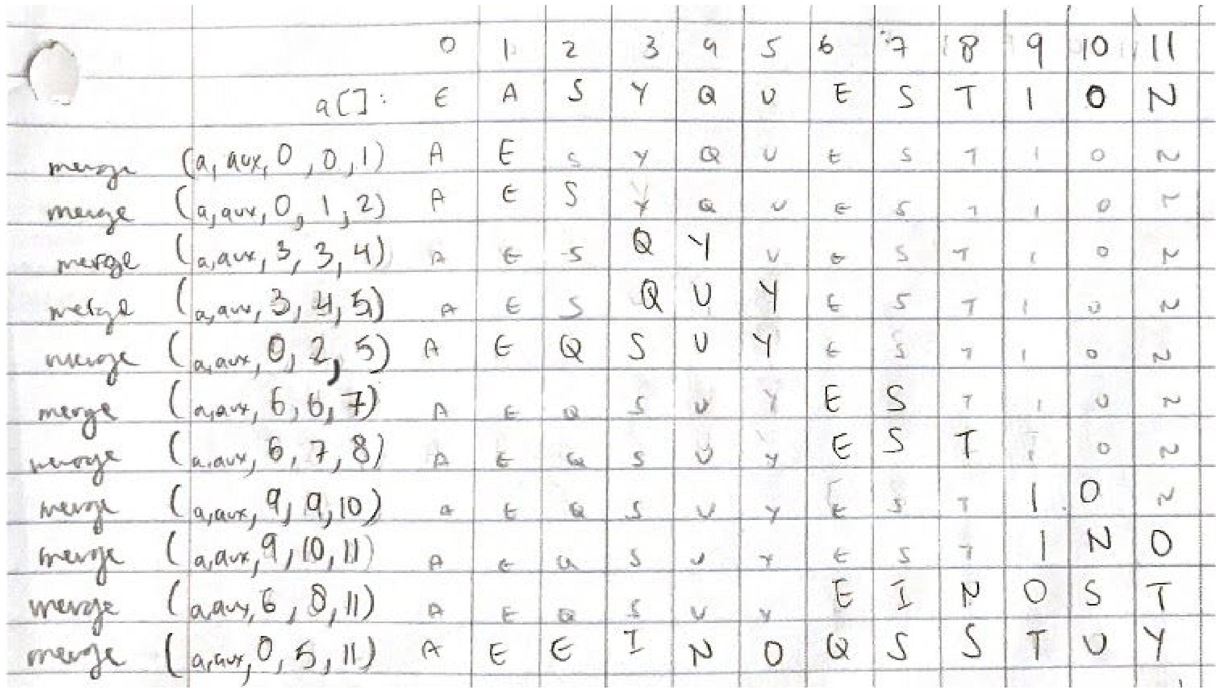
Catherine Kuntoro
CS 146
Professor Chandak
8 March 2020

Theory Answers for Homework 3

Question 4



		0	1	2	3	4	5	6	7	8	9	10	11
	a[]:	E	A	S	Y	Q	U	E	S	T	I	O	N
merge	(a, aux, 0, 0, 1)	A	E	S	Y	Q	U	E	S	T	I	O	N
merge	(a, aux, 0, 1, 2)	A	E	S	Y	Q	U	E	S	T	I	O	N
merge	(a, aux, 3, 3, 4)	A	E	S	Q	Y	U	E	S	T	I	O	N
merge	(a, aux, 3, 4, 5)	A	E	S	Q	U	Y	E	S	T	I	O	N
merge	(a, aux, 0, 2, 5)	A	E	Q	S	U	Y	E	S	T	I	O	N
merge	(a, aux, 6, 6, 7)	A	E	Q	S	U	Y	E	S	T	I	O	N
merge	(a, aux, 6, 7, 8)	A	E	Q	S	U	Y	E	S	T	I	O	N
merge	(a, aux, 9, 9, 10)	A	E	Q	S	U	Y	E	S	T	I	O	N
merge	(a, aux, 9, 10, 11)	A	E	Q	S	U	Y	E	S	T	I	N	O
merge	(a, aux, 6, 8, 11)	A	E	Q	S	U	Y	E	I	N	O	S	T
merge	(a, aux, 0, 5, 11)	A	E	E	I	N	O	Q	S	S	T	U	Y

Question 5

Assuming that the pivot is the very first element of the array, quick sort will make a total of $\sum_{i=1}^{n} (n - i)$ comparisons, in which the variable 'n' is the size of the array. For example, given the array [5, 5, 5, 5] with the first '5' as the pivot, the pivot will check every element to its right if it is bigger than itself. That means, the pivot will compare itself with three other elements. Then, the pivot moves on to the second element as its pivot, and the same process repeats, with two comparisons this time. Then, the pivot will move to the third '5', in which it will do only one comparison. Once the pivot reaches the last '5', the sorting has been completed. Therefore, with this given array, the quick sort performs 3+2+1+0 = 6 comparisons, which corresponds to the equation: $\sum_{i=1}^{n} (n - i)$, where in this example, n equates to four since the size of the array is four.

Question 6
The values of the remove the max operation, starting from the first asterisk:
R, R, P, T, Y, O, I, U, Q, I, E, U

Question 7
First, both methods of construction would need to prepare the given list of values by turning it into a max heap, usually, and there are two methods of such heap construction.

The first method of heap construction is proceeding from left to right through the heap by using the swim() method. It is from left to right since the heapify process starts from the leftmost child node. So, the child node that is to the left of the scanning pointer would need to swim() up to fix any heap violations on its way up. The swim() rearranges the heap by having the child look up to its parent's value, so if the child node's value is greater than the parent's a value exchange would occur. Thus, every item starting from the leftmost item would need to be swimmed up, therefore left to right heap construction has a O(Nlog(N)) time complexity.

The second method of heap construction is proceeding from right to left through the heap by using the sink() method. It is from right to left since the heapify process starts from the rightmost parental node. So, the scanning pointer that is to the right of its child node will have to sink() down to fix any heap violations on its way down. The sink() rearranges the heap by having the current node look down to its children nodes' value, and if the current node's value is lesser than the greatest value of the children node's, it would swap values with the greatest child value and thus sink down. In comparison to using the swim() method, heap construction from right to left has a time complexity of O(N).