Homework #6 B09705039 劉惟恩

Question 1 sol:

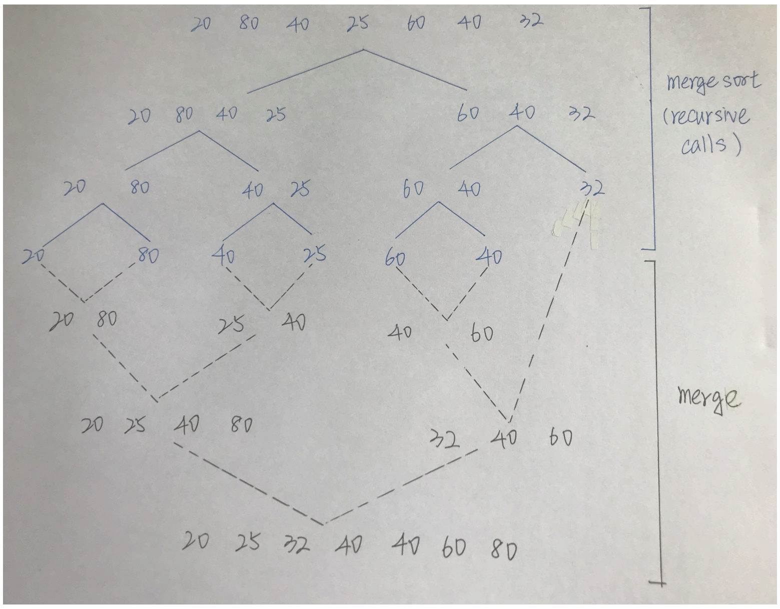
a. Selection sort :
step1: The largest number in the unsorted group is 80, so we swap 80 and 32 to the end of the array => the array becomes : 20 32 40 25 60 40 | 80 ("|" splits the sorted group from the unsorted)
step2: We found the largest number in the unsorted group again which is 60, so we swap 60 and 40 to the first element of the sorted group => the array becomes : 20 32 40 25 40 | 60 80
step3: We found the largest number in the unsorted group again which is 40, so we swap 40 and 40 to the first element of the sorted group => the array becomes : 20 32 40 25 | 40 60 80
step4: We found the largest number in the unsorted group again which is 40, so we swap 40 and 25 to the first element of the sorted group => the array becomes : 20 32 25 | 40 40 60 80
step5: We found the largest number in the unsorted group again which is 32, so we swap 32 and 25 to the first element of the sorted group => the array becomes : 20 25 | 32 40 40 60 80
step6: We found the largest number in the unsorted group again which is 25, so we swap 25 and 25 to the first element of the sorted group => the array becomes : 20 | 25 32 40 40 60 80
step7: We reach the last number in the sorted group, which means the selection sort is finished and the numbers are sorted. => the array becomes : 20 25 32 40 40 60 80

b. Insertion sort :
step1: Copy 80 and compare it with the sorted group in front of it, since 80 > 20 we insert it after 20. => the array becomes : 20 80 | 40 25 60 40 32
step2: Copy 40 and compare it with the sorted group in front of it, since 80 > 40 > 20 we insert it after 20 and in front of 80 by shifting 80. => the array becomes : 20 40 80 | 25 60 40 32
step3: Copy 25 and compare it with the sorted group in front of it, since 40 > 25 > 20 we insert it after 20 and in front of 40 by shifting 40 and 80. => the array becomes : 20 25 40 80 | 60 40 32
step4: Copy 60 and compare it with the sorted group in front of it, since 80 > 60 > 40 we insert it after 40 and in front of 80 by shifting 80. => the array becomes : 20 25 40 60 80 | 40 32
step5: Copy 40 and compare it with the sorted group in front of it, since 60 > 40 = 40 we insert it after 40 and in front of 60 by shifting 60 and 80, to maintain the array stable.
=> the array becomes : 20 25 40 40 60 80 | 32
step6: Copy 32 and compare it with the sorted group in front of it, since 40 > 32 > 25 we insert it after 25 and in front of 40 by shifting 40, 40, 60 and 80. End of insertion sort numbers are sorted.
=> the array becomes : 20 25 32 40 40 60 80

c. Bubble sort :
step1: Compare 20 and 80, since 20 < 80, do nothing.
step2: Compare 80 and 40, since 40 < 80, swap 40 and 80.
=> the array becomes : 20 40 80 25 60 40 32
step3: Compare 80 and 25, since 25 < 80, swap 80 and 25.
=> the array becomes : 20 40 25 80 60 40 32
step4: Compare 80 and 60, since 60 < 80, swap 80 and 60.
=> the array becomes : 20 40 25 60 80 40 32
step5: Compare 80 and 40, since 40 < 80, swap 80 and 40.
=> the array becomes : 20 40 25 60 40 80 32
step6: Compare 80 and 32, since 32 < 80, swap 80 and 32.
=> the array becomes : 20 40 25 60 40 32 80
step7: Compare 20 and 40, since 20 < 40, do nothing.
step8: Compare 40 and 25, since 25 < 40, swap 40 and 25.
=> the array becomes : 20 25 40 60 40 32 80
step9: Compare 40 and 60, since 40 < 60, do nothing.
step10: Compare 60 and 40, since 40 < 60, swap 60 and 40.
=> the array becomes : 20 25 40 40 60 32 80
step11: Compare 32 and 60, since 32 < 60, swap 32 and 60.
=> the array becomes : 20 25 40 40 32 60 80
step12 ~ step14: Compare a and b, since a < b, do nothing.
step16: Compare 40 and 32, since 32 < 40, swap 32 and 40.
=> the array becomes : 20 25 40 32 40 60 80
step17 ~ step18: Compare a and b, since a < b, do nothing.

step19: Compare 40 and 32, since 32 < 40, swap 32 and 40.
=> the array becomes : 20 25 32 40 40 60 80
step20 ~ step31: Compare a and b, since a < b, do nothing. Array is sorted end bubble sort.

d. Merge sort :



e. Quick sort :
step1: Choose pivot by median-of-three pivot method. => 20 < 25 < 32 => choose 25 as pivot
=> the array becomes : **20** 80 40 40 60 **25** **32** <*stage : quicksort recursion*>
step2: Index from left start from 80, index from right start from 60, do nothing. Swap 25 and 80.
=> the array becomes : 20 25 40 40 60 80 32 <*stage : partition*>
step3: Choose pivot of the bigger group by median-of-three pivot method. => 60 < 40 < 32 => choose 40 as pivot => the array becomes : 20 25 **32** 40 80 **40** **60** <*stage : quicksort recursion*>
step4: Index from left start from 40, index from right start from 80, do nothing. Swap 40 and 80.
=> the array becomes : 20 25 32 40 40 80 60 <*stage : partition*>
step5: Bubble sort smaller group 32 40. => the array becomes : 20 25 32 40 40 80 60
<*stage : quicksort recursion*>
step6: Bubble sort bigger group 80 60. => the array becomes : 20 25 32 40 40 60 80
<*stage : quicksort recursion*>

Question 2 sol:

a. Worst case:
(30 - 1) + (30 - 2) + (30-3) + …… + 1 = (29 + 1) * 29 / 2 = 15 * 29 = 435 comparisons

b. Best case:
(30 - 1) = 29 comparisons

Question 3 sol:

Time complexity calculation:
1) selection sort : O(n^2) => comparisons : (n - 1) + (n - 2) + …… + 1 = (n^2 - n) / 2 => O(n^2)
2) bubble sort : O(n^2) => comparisons : (n - 1) + (n - 2) + …… + 1 = (n^2 - n) / 2 => O(n^2)
3) insertion sort : O(n^2) => comparisons : 1 + 2 + …… + (n - 1) = (n^2 - n) / 2 => O(n^2)
4) merge sort : O(nlog(n)) => same level comparisons : n - 1, recursive level = log(n) => O(nlog(n))
5) quick sort : average case O(nlog(n)), worst case O(n^2)
=> same level comparisons : n - 1, recursive level = log(n) => O(nlog(n)) (average case)
=> same level comparisons : n - 1, recursive level = n => O(n^2) (worst case)

Since O() only represents the time complexity as n is big enough, we have to split the performance in to two situations. (We assume that the quick sort below isn't the worst case.) :

Large data performance : quick sort = merge sort > selection sort = bubble sort = insertion sort
Small data performance : selection sort = bubble sort = insertion sort > quick sort = merge sort