Maria Corella
4418914

Computer Science I – Exercise: Sorting

Before starting the exercise, go through the full slides, simulations, codes, and run time analysis. Then start doing the exercise.

1)Show the contents of the array below being sorted using Insertion Sort at the end of each loop iteration.

Initial	2	8	3	6	5	1	4	7
	2	3	8	6	5	1	4	7
	2	3	6	8	5	1	4	7
	2	3	5	6	8	1	4	7
	1	2	3	5	6	8	4	7
	1	2	3	4	5	6	8	7
	1	2	3	4	5	6	7	8
Sorted	1	2	3	4	5	6	7	8

2) Show the contents of the array below being sorted using Selection Sort at the end of each loop iteration. As shown in class, please run the algorithm by placing the smallest item in place first.

Initial	6	2	8	1	3	7	5	4
①	1	2	8	6	3	7	5	4
②	1	2	8	6	3	7	5	4
③	1	2	3	6	8	7	5	4
④	1	2	3	4	8	7	5	6
⑤	1	2	3	4	5	7	8	6
⑥	1	2	3	4	5	6	8	7
Sorted	1	2	3	4	5	6	7	8

3) Show the contents of the array below being sorted using Bubble Sort at the end of each loop iteration. As shown in class, please run the algorithm by placing the largest item in place first.

Initial	4	2	6	5	7	1	8	3
	2	4	5	6	1	7	3	8
	2	4	5	1	6	3	7	8
	2	4	1	5	3	6	7	8
	2	1	4	3	5	6	7	8
	1	2	3	4	5	6	7	8
	1	2	3	4	5	6	7	8
Sorted	1	2	3	4	5	6	7	8

4) When Merge Sort is run on an array of size 8, the merge function gets called 7 times. Consider running Merge Sort on the array below. What would the contents of the array be right before the 7th call to the Merge function?

Initial	7	2	1	5	8	3	4	6
Before 7th Merge	1	2	5	7	3	4	6	8

7 2 1 5 8 3 4 6
7 2 1 5 8 3 4 6
7 2 1 5 8 3 4 6

① 2 7 ② 1 5 ③ 3 8 ④ 4 6
⑤ 1 2 5 7 ⑥ 3 4 6 8
⑦ 1 2 3 4 5 6 7 8

5) Show the result of running Partition (as shown in class on Friday) on the array below using the leftmost element as the pivot element. Show what the array looks like after each swap.

pivot

Initial	5	2	1	7	8	3	4	6
	5	2	1	4	8	3	7	6
	5	2	1	4	3	8	7	6
After Partition	4	2	1	5	3	8	7	6

6) Show the contents of the array below after each merge occurs in the process of Merge-Sorting the array below:

Initial	3	6	8	1	7	4	5	2
	3	6	8	1	7	4	5	2
	3	6	1	8	7	4	5	2
	3	6	1	8	4	7	5	2
	3	6	1	8	4	7	2	5
	1	3	6	8	4	7	2	5
	1	3	6	8	2	4	5	7
Last	1	2	3	4	5	6	7	8

7) Here is the code for the partition function (used by Quick Sort). Explain the purpose of each line of code.

```
int partition(int* vals, int low, int high) {

  int lowpos = low; => sets the index for the low position to the leftmost index
  low++; => increases low to start comparing

  while (low <= high) { => loops => loops while low is less than high
                          loop until a number bigger than lowpos is found
    while (low <= high && vals[low] <= vals[lowpos]) low++;
    while (high >= low && vals[high] > vals[lowpos]) high--;
                          loop until a number smaller than lowpos is found
    if (low < high)
      swap(&vals[low], &vals[high]); => swap value between low and high
  }

  swap(&vals[lowpos], &vals[high]); => swap the pivot element into proper location
  return high; => return index of partition
}
```

8) Explain, why in worst case scenario the quick sort algorithm runs more slowly than Merge Sort algorithm. It doesn't halve the array in each step. If smallest or lowest element is chosen it runs n times

9) In practice, quick sort runs slightly faster than Merge Sort. This is because the partition function can be run "in place" while the merge function can not. More clearly explain what it means to run the partition function "in place". In place means you modify the original array. Don't need to allocate memory / create a new one.

10. You are trying to write a code for selection sort and you come-up with the following code. However, there is a bug in the code. Identify that bug and explain why that is a bug and edit that part of the code to correct it. Later, analyze the run-time of the updated code:

```
void selectionSort(int arr[], int n)
{
    int i, j, min_idx, temp;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++)
    {
        min_idx = i;
        for (j = 0; j < n; j++)
          if (arr[j] < arr[min_idx])
            min_idx = j;


       temp = arr[i];
        arr[i] = arr[min_idx];
        arr[min_idx] = temp;

     }
}
```

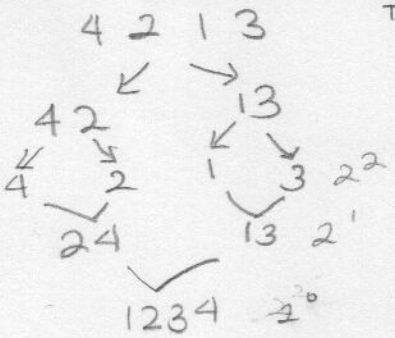
j should be j = i+1, otherwise it will always compare to arr[0] which is the smallest number. You want to start comparing at the index after the one you are on

Goes through array n-1 times and goes through n-i elements each time. i being iteration number.

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

Ever iteration, the function has one less item to iterate through because it has been moved to its correct position.

11) Explain the steps to come-up with the recurrence relation for merge sort and solve the recurrence relation to get the run-time of merge sort.



$T(n) = c + T(n/2) + T(n/2) + O(n)$

$T(n) = 2T(n/2) + c + O(n)$

$T(n) = 2T(n/2) + n$

$= 4T(n/4) + 2n$

$= 8T(n/8) + 3n$

$T(n) = 2^k T(n/2^k) + kn$

$= nT(1) + n\log_2 n$

$O(n * \log n)$

$T(1) = 1$

$T(n/2) = 2T(n/4) + n/2$

$T(n/4) = 2T(n/8) + n/4$

$T(n/8) = 2T(n/16) + n/8$

$\frac{n}{2^k} = 1 \quad k = \log_2 n$

$n = 2^k$