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	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
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.

2 is already sorted →

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

repeats 2 more times even though sorted →

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

↑ split

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.

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	2	1	4	3	5	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
1st	3	6	1	8	7	4	5	2
	1	3	6	8	7	4	5	2
2nd	1	3	6	8	4	7	5	2
	1	3	6	8	4	7	2	5
	1	3	6	8	2	4	5	7
	1	2	3	4	5	6	7	8
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; Set Pivot to first element
  low++; Set low to next element

  while (low <= high) { loop until low & high cross

    while (low <= high && vals[low] <= vals[lowpos]) low++; traverse until
    while (high >= low && vals[high] > vals[lowpos]) high--; Point where low
                                                              and high swap

    if (low < high)
      swap(&vals[low], &vals[high]); Swap lower val with higher val
  }

  swap(&vals[lowpos], &vals[high]); Swap Pivot to correct location
  return high; Return index of correct Pivot location
}
```

8) Explain, why in worst case scenario the quick sort algorithm runs more slowly than Merge Sort

merge sort always splits the array in half, but
quick sort is dependant upon left and right values of the
lowest & highest elements for run time.

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".

while merge sort repeatedly splits the array in half $(\frac{n}{2})$ for each
recursive call, the partition function in quick sort operates in terms
of $O(n)$.

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++)    Should be j = i+1;
          if (arr[j] < arr[min_idx])
            min_idx = j;

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

    }
}
```

if j starts at 0, then that is not near the min_idx. Since min_idx should be the minimum index, and j should start from the next index before iteration.

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.

There are 2 recursive calls (for each half of the array being sorted) This is expressed as $T(\frac{n}{2}) + T(\frac{n}{2})$. From here, use the iteration and substitution method to get the runtime of mergeSort.