# 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 | 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 |

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 |

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 | 3 | 2 | 1 | 4 | 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 |
| | 3 | 6 | 1 | 8 | 7 | 4 | 5 | 2 |
| | 1 | 3 | 6 | 8 | 7 | 4 | 5 | 2 |
| | 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;where pivot is
  low++;pointer after pivot

  while (low <= high) {Go until they cross

    while (low <= high && vals[low] <= vals[lowpos]) low++;move low till value
    while (high >= low && vals[high] > vals[lowpos]) high--; is too big & vice versa

    if (low < high) they haven't crossed so swap them
      swap(&vals[low], &vals[high]);
  }

  swap(&vals[lowpos], &vals[high]);
  return high; Swap partition to correct value
}
```

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

The emount of work does not get cut in half each recursive cell

Worst case $O(n^2)$

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".

Requires no extra space to function

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 = i+1

no need to check index 0

$T = n + (n+1) + (n+2) + (n+3) . + 3 + 2 + 1$

adding 1 to n each iteration

$n(n+1)/2$

resulting in an $O(n^2)$ algorithm

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.

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

Base. $T(1) = 1$

each time the array is split $n/2$
each time the array is merged
$O(n)$

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

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

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

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

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

PATTERN

$n/2^k = 1$

$n = 2^k$

$k = \log_2 n$

$2^{\log_2 N} T(1) + (\log_2 n) n$

$T(n) = n + n \log n > O(n^* \log n)$ time