CS3230 Homework 1

Name: Aadit Kamat

1.

Let us call the value extracted from the heap at line 6 of the algorithm for the mth iteration of the loop v.

1.1. The statement is the following invariant: given that m-1 elements of the array B are sorted correctly, v is sorted correctly into the mth position in array B i.e. it is indeed the mth smallest element in array B $\forall \mathrm{m}, 1 <= \mathrm{m} <= \mathrm{n}$

1.2. Proof by contradiction:

1.2.1. Let us assume that assume that the statement in 1.1. is wrong i.e. m – 1 elements of the array B are sorted correctly but the algorithm does not sort the correct element into the mth position of array B.

1.2.2. This means that there is a smaller element than v which can be inserted into the mth position of array B. Let us call that element x.

1.2.3. x cannot be a value in the heap S since the extractMin() gets the minimum of all elements in the heap.

1.2.4. x must be one of the values remaining in array A after insertion into the heap at step 5 of the algorithm i.e. x $\in \{\mathrm{A}[(\mathrm{m}+1)+\mathrm{k}] \ldots . \mathrm{A}[\mathrm{n}]\}$

1.2.5. This contradicts the fact that the array A is k sorted since the mth smallest value is not in positions A[1], …, A[m + k]

1.2.6. Hence, our assumption step is wrong i.e. if the m – 1 elements of the array B are sorted correctly, then v must be the mth smallest element in the array B.

1.3. Basis: First element in B is the minimum element in array A

Hypothesis: All the m – 1 elements in array B are sorted correctly

Induction step: The value extracted from the heap S is sorted correctly into the mth position in array B

The link between the hypothesis and the induction step is established in 1.1 and proved in 1.2. What remains to be proven is the basis.

Proof of the basis:

1.3.1. Let u be the smallest element in B

1.3.2. Let us suppose that basis is not true i.e. there is an element smaller than u in array A that can be inserted in B. Let us call that element w.

1.3.3. Since the extractMin() gets the minimum of the all the elements in the heap S, the element w must be one of the values remaining in A after insertion of an element into heap S in the first iteration of the loop at step 4 of the algorithm i.e. w $\in$ {A[k + 2] … A[n]}

1.3.4. However, this contradicts the fact that input array A is k sorted (the minimum element in array A is in one of the positions A[1] … A[k + 1] which consists of all the elements currently in the heap S).

1.3.5. This means our assumption at step 2 is wrong and hence the basis is proved true.

2.

2.1. The 5 inversions are: (8, 6), (6, 1), (8, 1), (3,1) & (2, 1)

2.2. A reverse sorted array: {n, n-1, n-2, …. 1} would have the most inversions.

It has $\sum_{i=1}^{i=n-1} n - i$ inversions i.e. $\frac{n(n-1)}{2}$

2.3. countIndividualInversions(arr, from, to)

```
 if from == to
   return 0

between ← from + (to – from) / 2
return countIndividualInversions(arr, from, between) + countIndividualInversions(arr, between + 1, to) +
        countMergedInversions(arr, from, between, between + 1, to)

countMergedInversions(arr, fromFirst, toFirst, fromSecond, toSecond)
    i ← fromFirst
    j ← fromSecond
    k ← 0
   ctr ← 0
   while i <= toFirst and j <= toSecond
       if arr[i] <= arr[j] then
           result[k] ← arr[i]
           i ← i + 1
      else
           result[k] ← arr[j]
           j ← j + 1
           ctr ← toFirst – j + 1
      k ← k + 1

   if i <= toFirst
     while i <= toFirst
         result[k] ← arr[i]
         i ← i + 1
         k ← k + 1
  else
     while j <= toSecond:
       result[k] ← arr[j]
```

```
        j ← j + 1
        k ← k + 1
    return ctr
```

The main difference introduced in the merge sort algorithm is modifying the merge step by adding k to the ctr if an element (say l) in the left half of the array is greater than an element (say r) in the right half of the array and there are k – 1 elements after l (since the two halves of the array are both sorted, all the elements after l will also be greater than r and thus lead to inversions).