CS/ECE 374 P13

Jiawei Tang, Pengxu Zheng, Junquan Chen

TOTAL POINTS

100 / 100

QUESTION 1

1 Problem 13.A. **30 / 30**

✓ **- 0 pts** Correct

QUESTION 2

2 Problem 13.B. **50 / 50**

✓ **- 0 pts** Correct

QUESTION 3

3 13.C. **20 / 20**

✓ **- 0 pts** Correct

HW Solution

CS/ECE 374: Algorithms & Models of Computation, Spring 2019 Version: **1.0**

Submitted by:

- ≪**Jiawei Tang**≫: ≪**jiaweit2**≫
- ≪**Junquan Chen**≫: ≪**junquan2**≫
- ≪**Pengxu Zheng**≫: ≪**pzheng5**≫

13

Solution:

```
i = 0
while i < n
    j = 0
    while j < n - u
        sortBlock(j)
        j += u/2
    i += u/2
```

A)

Consider the worst case where an array is sorted in decreasing order. In such algorithm, we will be calling sortBlock at the very first element in A[i...,i+u]. Then we try to shift the input of sortBlock by $\left\lfloor \frac{u}{2} \right\rfloor$. Now $u/2$ of max values in this array are carried with the shift. In this case, $\left\lfloor \frac{u}{2} \right\rfloor$ elements will be sorted at the end of the array within one iteration of the array. Each call of sortBlock sort $u/2$ elements so each iteration calls sortBlock $n/(u/2) = 2n/u$ times. To completely sort this array we need $2n/u$ iteration. Therefore, with $2n/u$ calls for each of 2n/u iterations. This algorithms is $O((n/u)^2)$ when calling sortBlock takes $O(1)$ time.

1 Problem 13.A. **30 / 30**

✓ **- 0 pts** **Correct**

B)

Referenced from Gargi Deb's group

```
k = 1
while(k <= n):
    iEnd = min(i+u, n/2)
    jEnd = min(j+u, n)
    copy(A[i...iEnd], W[k...(iEnd-i)])
    begin = k
    k = k + (iEnd-1) + 1
    copy(A[j...jEnd], W[k...(jEnd-j)])
    end = k + (jEnd-j)
    ret = bMerge(W[begin...(k-1)], W[k...end])
    addup = k - begin
    i = i + ret
    j = j + (addup-ret)
    k = 1
    while(k <= n):
        end = min(k + u, n)
        copy(W[k...end], A[k...end])
        k = end +1
```

Notice that this case is very similar to the merge step of mergesort.We are given an array which are divided into two sorted parts. Left and right parts are both sorted. Then we need to merge the sorted halves into the work array W. Note that the lower elements of the lowest of both will be placed in the left portion after calling bMerge. Also, we know the number of elements from each portion contains this lower portion and we could use that to increment each half by the exact amount. We increment k value equal to the first portion and repeat till the W array is full. We then copy back the array to A. In the worst case, we have to take care of all elements of the array, and do the copy operation. Note that copy (either directions) takes O(1) time.

2 Problem 13.B. 50 / 50

✓ - 0 pts Correct

C)

```
def mergesort(A, start, end):
 if start ≠ end:
    mid = ⌊(start+end)/2⌋
    mergesort(A,start, mid)
    mergesort(A,mid+1, end)
    sortbyParts(A,start,mid+1, end)
```

We are basically implementing mergesort so we need to use the function we implemented in b). Lets define the pseudo function in b) as sortbyParts(A,i,j,n). Knowing the two ending point: start and end. divide by finding the number of position midway between start and end, by adding start and end and divide by 2(round down). We now can recursively call the mergesort function until we meet the base case. Then we can conquer by sorting the subarrarys and then combine by merging two sorted elements back to a single sorted subarray until it reaches the top one sorted array. In mergesort, the runtime is $O(nlog(n))$. At first the array is constantly being divided into halves and this process takes O(log(n)) time. Since sortbyparts take $O(n)$ times, so we have $O(nlog(n))$ as our final runtime.

3 13.C. **20 / 20**

✓ **- 0 pts** Correct