TCSS543A – Autumn 2020
Homework 1 (30 points)

Student Name – Siddharth Sheth
Student ID – 2076180
TCSS (Autumn -2020)

1. (**2+2+2+2+2 points**) Find and prove (e.g. using the Master Theorem or other method) tight bounds for the following recurrences (where *h* is a positive constant). To get full grade, you must correctly justify your answer (i.e. lucky guesses will not help):

(a) a=16 (a >=1)
b = 32 (b > 1)
So, here recurrences is in standard format to move ahead with Master Theorem.
c= $\log_b a = \log_{32} 16 = 4/5 = 0.8$
f(n) = $n/\sqrt[5]{n} + 1/\sqrt{n} = (n^{4/5} + n^{(-1/2)})$

Let's check First case now,

f(n) ϵ O $(n^{c-\epsilon})$ => $n^{4/5} + n^{(-1/2)}$ ϵ O $(n^{0.8-\epsilon})$

Now, let's apply Limit test

$\lim_{n\to\infty} \frac{f(n)}{(n^{0.8-\epsilon})} = \lim_{n\to\infty}$ $(n^{4/5} + n^{(-1/2)})/ (n^{0.8-\epsilon})$; here for any value of ϵ limit will tends to ∞ and not to 0. So, <u>case – 1 is not applicable here</u>.

Let's check second case now,

F(n) = Θ $(n^c ln^k n)$; k ≥ 0

Now, let's apply Limit test

$\lim_{n\to\infty} \frac{f(n)}{n^2 ln^k n} = \lim_{n\to\infty} (n^{4/5} + n^{(-1/2)}) / (n^{4/5} ln^k n)$ (' c = 4/5)

$= \lim_{n\to\infty} (n^{4/5} + n^{(-1/2)}) / n^{4/5}$ (For K = 0; $ln^k n = ln^0 n = 1$)

$= \lim_{n\to\infty} \frac{n^{\frac{4}{5}}}{n^{\frac{4}{5}}} + \frac{n^{-1/2}}{n^{4/5}}$

$= \lim_{n\to\infty} 1 + \frac{1}{n^{4/5}+n^{1/2}}$

= 1 (As 1 + 1/∞ = 1+ 0 = 1)

So, here case 2 works.

Answer: T(n) ϵ $\Theta(n^c\ ln^{k+1} n)$ (k =0, c = $\log_{32} 16$)

(b)

a = 16 (a ≥ 1)
b = 16 (b > 1)
So, here recurrences is in standard format to move ahead with Master Theorem.
c = $\log_b a = \log_{16} 16 = 1$
f(n) = $n^{\lg n}$

Let's check First case now,

f(n) ϵ O $(n^{c-\epsilon})$ => $n^{\lg n} \epsilon$ O $(n^{1-\epsilon})$

Now, let's apply Limit test

$\lim_{n\to\infty} \frac{f(n)}{(n^{c-\epsilon})} = \frac{\lim_{n\to\infty} n^{\lg n}}{n^{1-\epsilon}}$; so here limit will tends to infinity. So, <u>case-1 is not possible</u>.

Let's check second case now,

$f(n \epsilon \Theta(n^c)(log^k n)$
Now, let's apply Limit test,

$\lim_{n\to\infty} \frac{n^{\lg n}}{(n^c)(log^k n)} = \lim_{n\to\infty} \frac{n^{\lg n}}{(n^1)(log^k n)}$; if we take K=0, then also limit tends to infinity not to any constant. So, <u>case-2 is not applicable here.</u>

Let's check third case now,

$f(n) \epsilon\ \Omega(n^{c+\epsilon})$; $a\ f\left(\frac{n}{b}\right) \leq d\ f(n)$ where $0 \leq d < 1$
.Let's apply limit test here,

$\lim_{n\to\infty} \frac{n^{\lg n}}{n^{c+\epsilon}} = \lim_{n\to\infty} \frac{n^{\lg n}}{n^{1+\epsilon}}$; here limit tends to infinity. Now, let's check <u>regularity condition</u>,

$a\ f\left(\frac{n}{b}\right) \leq d\ f(n)$
$\therefore 16\ (\frac{n}{16})^{\lg n/16} \leq d(n^{\lg n})$
$\therefore (16)(\frac{n}{16})^{\lg n}\ (\frac{n}{16})^{\lg 1/16} \leq d(n^{\lg n})$
$\therefore (16)(1/16)^{\lg n}(n)^{\lg n}(\frac{n}{16})^{\lg 1/16} \leq d(n^{\lg n})$
$\therefore\ (16)(1/16)^{\lg n}(\frac{n}{16})^{-4} \leq d$ (as lg 1/16 = -4)
$\therefore (16)(\frac{1}{16^{\lg n}})(\frac{n}{16})^{-4} \leq d$
$\therefore (16)(\frac{1}{n^4})(\frac{n}{16})^{-4} \leq d\ (as\ 16^{\lg n} = 2^{\lg n^4} = n^4)$
$\therefore \frac{16^5}{n^8} \leq d$

Now, for n = 16, regularity condition will be $\frac{1}{16^3} \leq d < 1$ (as by definition 0 < d < 1).
So, <u>case -3 is applicable here.</u>

Answer: T(n) ϵ $\Theta(n^{\lg n})$

(c) a=38
b=22
So here recurrences is in standard format to move ahead with Master Theorem
c = $\log_b a$ = $\log_{22} 38$ = 1.17
f(n) = n($n^{1/6}$)

Let's check First case now,

f(n) ϵ O ($n^{c-\epsilon}$) => n($n^{1/6}$)ϵ O ($n^{1.17-\epsilon}$)

Now, let's apply Limit test

$\lim_{n\to\infty} \frac{n(n^{1/6})}{(n^{1.17-\epsilon})} = \lim_{n\to\infty} \frac{n^{7/6}}{n^{1.17-\epsilon}} = \lim_{n\to\infty} n^{-0.01+\epsilon}$

$\therefore \epsilon - 0.01 < 0$
$\therefore 0 < \epsilon < 0.01$
So, here for $\epsilon < 0.01$, limit tends to zero. So ,<u>case -1 is applicable here</u>.

Answer: T(n) ϵ O ($n^{c-\epsilon}$) (c = $\log_{22} 38$)

(d) a=6/5 (a >=1)
b = 7/6 (b > 1)
So, here recurrences is in standard format to move ahead with Master Theorem.
c= $\log_b a$ = $\log_{7/6} 6/5$ = 0.86
f(n) = n($n^{1/4}$)

Let's check First case now,

f(n) ϵ O ($n^{c-\epsilon}$)

$\lim_{n\to\infty} \frac{n(n^{1/4})}{n^{c-\epsilon}} = \lim_{n\to\infty} \frac{n^{5/4}}{n^{0.86-\epsilon}} = \lim_{n\to\infty} n^{1.25-0.86+\epsilon} = \lim_{n\to\infty} n^{0.39+\epsilon}$

Here, ϵ > 0; so n will increase and for any value of ϵ limit will tends to ∞ and not to 0. So, <u>case – 1 is not applicable here.</u>

Let's check second case now,

F(n) = Θ ($n^c ln^k$n); k $\geq$ 0

Now, let's apply Limit test

$\lim_{n\to\infty} \frac{n^{5/4}}{(n^{0.86})(log^k n)} = \lim_{n\to\infty} \frac{n^{0.39}}{(log^k n)}$

Let's have k = 0; so $log^k n$ = 1 and as n tends to ∞, limit here will tends to infinity not a constant. So, <u>case -2 is not applicable here.</u>

Let's check third case now,

$f(n) \epsilon\ \Omega(n^{c+\epsilon})$; $a\ f\left(\frac{n}{b}\right) \leq d\ f(n)$ where $0 \leq d < 1$

.Let's apply limit test here,

$\lim_{n\to\infty} \frac{n^{5/4}}{n^{0.86+\epsilon}} = \lim_{n\to\infty} n^{1.25-0.86-\epsilon} = \lim_{n\to\infty} n^{0.39-\epsilon}$

So, for $\epsilon < 0.39$ limit tends to ∞.

Now, let's check regularity condition

$a\ f\left(\frac{n}{b}\right) \leq d\ f(n)$
$\therefore \frac{6}{5} \left(\frac{n\,6}{7}\right)^{5/4} \leq$ d $(n^{5/4})$
$\therefore \frac{6}{5} \left(\frac{6}{7}\right)^{5/4} (n)^{5/4} \leq$ d $(n^{5/4})$
$\therefore \frac{6}{5} \left(\frac{6}{7}\right)^{5/4} \leq$ d
$\therefore 1.2 \times 0.82 \leq$ d
$\therefore 0.97 \leq$ d
So, for $0.97 \leq$ d < 1, regularity condition gets satisfied here.
So, <u>case -3 is applicable here</u>.

Answer: $\boldsymbol{T(n) \epsilon\ \Omega(n^{c+\epsilon})}$ **;** $\boldsymbol{a\ f\left(\frac{n}{b}\right) \leq d\ f(n)}$ **(c =** $\boldsymbol{\log_{7/6} 6/5}$**;** $\boldsymbol{0 < \epsilon < 0.39}$**; 0.97 ≤ d < 1)**

(e) a = 3/5 (a < 1)
As, a < 1, Master's theorem can't be applied to this recurrence.

Let's try to find tight bound by recurrence method.

T(n) = 3/5 T (3n /5) + h/n

Let's find T(3n/5).
T(3n/5) = 3/5 T (3/5)(3n/5) + 5h/3n

Let's substitute value of T(3n/5) in the given recurrence:

So, T(n) = 3/5 [3/5 T((3n/5)(3/5)) + 5h/ 3n] + h/n
T(n) = $(3/5)^2$ [T $(3/5)^2\ n$] + 6h/4n
T(n) = $(3/5)^k$ [T $(3/5)^k\ n$] + kh/n

T(1)= 1, and let's assume n = (5/3)k
$\log_{5/3} n = k$ (taking log both sides)

Now, let's put value of K,

T(n) = $(3/5)^{\log_{5/3} n}$ [T $(3/5)^{\log_{5/3} n}$ n] + ($\log_{5/3} n$)h/n

$\therefore T\ (n)\ \epsilon\ (\frac{\log n}{n})$

Now, let's prove it by induction

T(n) = 3/5 T (3n /5) + h/n

= 3/5 $\frac{(log\ 3n/5)}{3n/5}$ + $\frac{h}{\frac{3n}{5}}$

= $\frac{log\ 3n/5}{n}$ + 5h/3n

= log n/ n + 1/n

= 1/n (log n + 1)

$\therefore \boldsymbol{T(n)\ \epsilon\ \frac{logn}{n}}$

2. (**10 points**) Kleinberg and Tardos: Chapter 2, Exercise 6.

Consider the following basic problem. You're given an array A consisting of n integers [1], [2], …, $A[n]$. You'd like to output a two-dimensional n-by- array B in which $B[i, j]$ (for $< j$) contains the sum of array entries $[i]$ through $[j]$—that is, the sum $[i] + A[i + 1] + \cdots + A[j]$. (The value of array entry $[i, j]$ is left unspecified whenever $i \geq j$, so it doesn't matter what is output for these values.)
Here's a simple algorithm to solve this problem.

```
for i = 1, 2, …, n
      for j = i + 1, i + 2, …, n
            add up array entries A[i] through A[j] store the
            result in B[i,j]
      end for
end for
```

(a) For some function f that you should choose, give a bound of the form $(f(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).

Answer:

Here, two for loops are given and both are nested one. Also, there are multiple summation operation is performed to get the output. So, let's say i = 1, so this addition operation will get executed n times for n elements. So, for addition operations O(n). Now, for same scenario outer loop will run for n times and inner loop will also run n times as it's nested one. So, we need to multiply while calculating loops and which will result as $O(n^2)$. Now, to get final output we need to multiply both so we can say total time will be **$f(n) \in O(n^3)$**

(b) For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)

Answer:

For i=1; j=i+1=2. So, inner loop will get executed double time than the outer loop. So, let's say outer loop gets executed n times, so inner loop will get executed for 2n times. Also, the addition in inner loop will take place at least the input of outer loop, so it will get calculated at least for n times. In total this gives us $2n^3$ operations that are at least required. Hence, f(n) is also in $\Omega(n^3)$ and therefor **$f(n) \in \Omega(n^3)$.**

(c) Although the algorithm you analyzed in parts (a) and (b) is the most natural way to solve the problem—after all, it just iterates through the relevant entries of the array B, filling in a value for each—it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $(f(n))$, where $\lim_{n\to\infty} (n)/(f(n)) = 0$.

NB: to get full points here, you must *justify* (i.e. *prove*) your calculations and statements with the limit test and/or by explicitly computing the summations whenever necessary.

Answer:

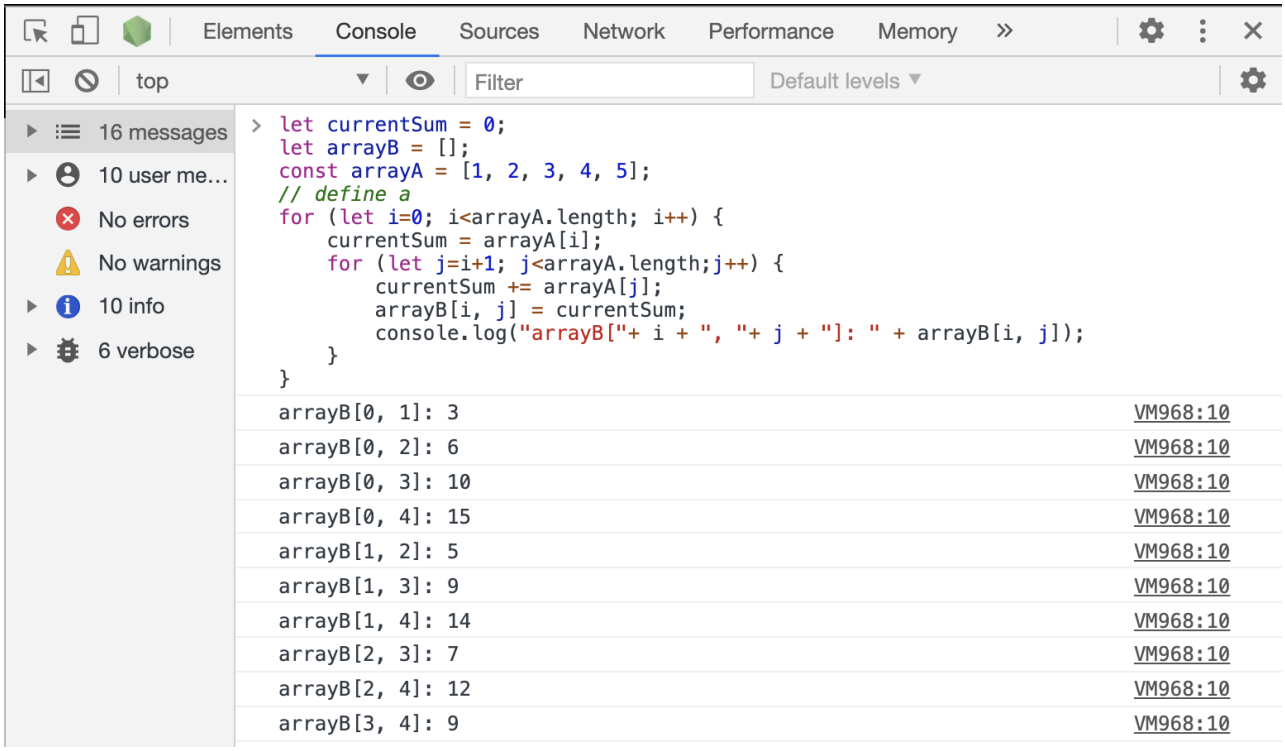
Here, ask is to get better running time. In the problem two nested loops and few addition operations are there. So, clearly we need to make addition operation optimal as two nested will run in each and every scenario.

So, considering this we can write algorithm as shown below –

```
let currentSum = 0;
let arrayB = [];
const arrayA = [1, 2, 3, 4, 5, ....n];

for (let i=0; i<arrayA.length; i++) {
   currentSum = arrayA[i];
   for (let j=i+1; j<arrayA.length;j++) {
      currentSum += arrayA[j];
      arrayB[i, j] = currentSum;                    //Storing the output
      console.log("arrayB["+ i + ", "+ j + "]: " + arrayB[i, j]);       //printing result on console
   }
}
```

I ran this algorithm on console and please find output screens-shot of the same below:



Now, let's find out why above mentioned loop is efficient.
Outer loop and inner loop will run for n times respectively. And as both loops are nested we can multiply to get O(n^2). And, inner addition operation will be linear so O(1) for them.
So, g(n) ϵ $O(n^2)$

As calculated in above questions, f(n) = n^3 and g(n) = n^2. So, algorithm derived in 2(c) is more efficient. Same can be tested with limit test as shown below –

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{n^2}{n^3} = \lim_{n\to\infty} \frac{1}{n} = 0$$

Therefore, **g(n) ϵ O (f(n))**

3. (**10 points**) Express the asymptotic running time (Big-Theta) of the following algorithm as a function of the input length n. Explain and justify your analysis:

```
void kmx(long[] a) {
    int n = a.length; // express the running time as a function of this n
    if (n <= 1) {
        return;
    }
    int t = 1;
    while (t < n - t) { t += t;
    }
    for (int p = t; p > 0; p >>= 1) {  for (int j = 0; j
        < n - p; j++) {
            if ((j & p) == 0) {
                if (a[j + p] < a[j]) {
                    long w = a[j]; a[j] = a[j + p]; a[j + p] = w;
                }
            }
        }
        for (int q = t; q > p; q >>= 1) {  for (int j = 0; j
            < n - q; j++) {
                if ((j & p) == 0) {
                    if (a[j + q] < a[j + p]) {
                        long w = a[j + p]; a[j + p] = a[j + q]; a[j + q] = w;
                    }
                }
            }
        }
    }
}
```

Answer:
To get total runtime of the algorithm, we need to find out runtime of each iteration or each loop.

So, here following command will take same running time with every input.

```
void kmx(long[] a) {
    int n = a.length; // express the running time as a function of this n
    if (n <= 1) {
        return;
    }
```

Let's say time taken by above mentioned four lines to run is **d……(i)**

Now, let's calculate total runtime for while loop. So, here for while loop condition given is t < n – t => 2t < n. Also t gets double on every loop iteration. So, while loop can be written as while (2t < n).

Now, to calculate the run time of the while loop, for the same if we can calculate how many times we need to double the t, until t will be equal or greater than n ?

Lets take an example to solve this, t = 2 and n = 1000. So, answer of above question can be

rounded up as $\log_2 1000$.

So, considering this while loop will run for (log n -1) iterations.
Therefore, if while loop takes z time to complete one iteration then total time by while loop will be **z(lg n -1)…….(ii)**

Now, let's move onto next loop. In this for loop, p = t. Here, p > 0; so p ≤ 0 ∴ t ≤ 0.
For loop has condition p >>=1 so it mean that set p to itself shifted by one bit to the right.
By definition, shifting all of a number's bits to the left by 1 bit is equivalent to multiplying the number by 2 and similarly shifting all of a number's bits to the right by 1 bit is equivalent to dividing the number by 2.
Here, let's say loop will have p/w steps. W = 2^k. So, we can calculate total number of operations as mentioned below -
∴ $\frac{p}{2^0} + \frac{p}{2^1} + \ldots\ldots + \frac{p}{2^k}$
∴ $k = \log_2 n$
$So, for\ outer\ for\ loop\ f(n) \epsilon\ \Theta(\log n)$

Now, let's calculate runtime of the inner loop.
Here, t < n/2, as t = p
p < n/2
So, best case would be p ≥ $n/2$. As, j = n – p; inner loop will run ≤ $n/2$ times.

Let's find bound using Master's theorem.

$\lim_{n\to\infty} \frac{\frac{n}{2}}{n} = \lim_{n\to\infty} \frac{1}{2}$. So, here inner loop in in $\Omega(n)$.

Now, in worst case p = 1, so inner loop will run n-1 times (as j = n –p).
So, we can say that loop will run for n times and thus, inner loop has upper bound O(n).

Here, inner loop has upper bound (O(n)) and Lower bound ($\Omega(n)$) both. So, it must be having tight bound $\Theta(n)$. So, let's say inner loop takes a time for single iteration. So, first nested for loop will take **a(nlogn)** time to execute**……(iii)**

Now, let's calculate run time of the second nested loop. Here, best case would be p = t, but as q=t and q<p; loop won't get executed. So loop executing time will be O(1) constant time. In worst case p =1 and as q = t and q > p (q >1); so it will run for q/w (w =2^k). So as calculated in above steps it will be (log n). And inner most loop we just calculated which is $\Theta(n)$ = b(nlog n). So, to get the result of the whole nested loop we need to multiply the result = **b(n$(log\ n)^2$)……(iv)**

So, now let's do summation of running time we calculated above of each loop to get the total run time of the loop. (Summation of results of steps (i), (ii), (iii), (iv))
∴ $d + z(\lg n - 1) + a(n \log n) + b(\mathrm{n}(log\ n)^2))$
This gives us tight bound for the running time = **$\Theta\ (n(\log n)^2)$.**

Let's apply Limit test,

$$\lim_{n\to\infty} \frac{d + z(\lg n - 1) + a(n \log n) + b(\mathrm{n}(log\ n)^2))}{n(\log n)^2}$$

$$= \lim_{n\to\infty} \frac{d+z(\lg n)-z)+a(n \log n)+b(\mathrm{n}(log\ n)^2))}{n(\log n)^2}$$

$$= \lim_{n\to\infty} \frac{d}{n(\log n)^2} + \frac{z(\lg n)}{n(\log n)^2} + \frac{(-z)}{n(\log n)^2} + \frac{a\, n \log n}{n(\log n)^2} + \frac{b\, n\, (\log n^2)}{n(\log n)^2}$$

$$= \lim_{n\to\infty} \frac{d}{n(\log n)^2} + \lim_{n\to\infty} \frac{z(\lg n)}{n(\log n)^2} - \lim_{n\to\infty} \frac{(-z)}{n(\log n)^2} + \lim_{n\to\infty} \frac{a\, n \log n}{n(\log n)^2} + \lim_{n\to\infty} \frac{b\, n\, (\log n^2)}{n(\log n)^2}$$

$$= 0 + \lim_{n\to\infty} \frac{z}{n(\log n)} - 0 + \lim_{n\to\infty} \frac{a}{\log n} + \lim_{n\to\infty} b$$

= b (which is constant)

$\therefore \boldsymbol{f(n) \epsilon \Theta(n(\log n)^2)}$

4. (**bonus: 2+2+1 points**) Keith and Randy, who work at the UW Tacoma Copy Center, have a series of n tasks of copying and then binding documents. They only have one copy machine and one binding machine, and at most one document can be processed by each of these machines at any time, but they can process distinct tasks simultaneously, as long as every document only enters the binding machine after it has been copied.

Based on the number of pages of each document, Keith and Randy know that the times needed to copy and bind them are, respectively, $A_1, \ldots, A_n$ and $B_1, \ldots, B_n$. Our friends need to know the shortest time they can get all tasks done and which order they should be performed, and they ask you to help them.

To that end, prove the following property:

(a) The sequence of documents processed by either machine can be made the same as that of the other machine without loss of time.

By property (a), we can assume Keith and Randy will feed the documents to the binding machine in the same sequence they feed them to the copier. Let S denote that sequence and let X_i denote the inactive period of time for the binding machine immediately before Keith and Randy feed it the i-th document to start the binding. For instance, $X_1 = A_{S[1]}$, $X_2 = \max(A_{S[1]} + A_{S[2]} - B_{S[1]} - X_{S[1]}, 0)$, and in general $X_i = \max(\sum_{j=1}^{i} [j] - \sum_{j=1}^{i-1} [j] - \sum_{j=1}^{i-1}, 0)$, from where we get the total idle time resulting from sequence S, denoted $F(S)$, to be $F(S) := \sum_{i=1}^{n} X_i = \max_{1\le u\le n} K_u$ where $K_u := \sum_{j=1}^{u} [j] - \sum_{j=1}^{u-1} [j]$. Our task is thus to find a sequence S that minimizes $F(S)$, that is, $F(S) \le F(T)$ for any other sequence T.

Given any sequence S and an index j, let S' denote the sequence obtained from S by

interchanging $[j]$ and $[j + 1]$, that is, $S'[j] = S[j + 1]$, $S'[j + 1] = S[j]$, and $S'[i] = S[i]$ for all $i \notin \{j, j + 1\}$. Also, let $K' := \sum_{j=1}^{u} A_{S'[j]} - \sum_{j=1}^{u-1} B_{S'[j]}$.

(b) Prove that an optimal sequence is done by the following rule: document $S[j]$ precedes $S[j + 1]$ iff $\max(K_j, K_{j+1}) < \max(K'_j, K'_{j+1})$.

(c) Design an algorithm that will find the shortest time that Keith and Randy can complete their copying and binding jobs, and also suggests an ordering of tasks that attains that shortest time.

Answer:

For (c) :: We can solve this by Work Conserving Scheduler Algorithm.

Ref: Discussed problems and possible approach with study group and fellow students. Also referred class slides, notes and textbooks/handbooks suggested as per syllabus.