TCSS 543 Homework #1

1. *Find and prove tight bounds for the following recurrences:*

(a) $T(n) = \begin{cases} c & n < 16 \\ 8\,T\left(\frac{n}{16}\right) + \frac{n}{\sqrt[4]{n}} + \frac{1}{\sqrt{n}} & n \geq 16 \end{cases}$

By Master's Theorem: $a = 8 \quad b = 16 \quad c = \log_b a = \log_{16} 8 = 0.75$

$f(n) = \frac{n}{\sqrt[4]{n}} + \frac{1}{\sqrt{n}}$

$= n^{1-0.25} + n^{-0.5}$

$= n^{0.75} + n^{-0.5} \in \begin{cases} O(n^{c-\varepsilon})? \Rightarrow T(n) \in \theta(n^{0.75}) & * Case\ 1 \quad \times \\ \theta(n^c \log^k n)? \Rightarrow T(n) \in \theta(n^{0.75} \log^{k+1} n) & * Case\ 2 \quad \surd \\ \Omega\,(n^{c+\varepsilon})? \Rightarrow T(n) \in \theta\big(f(n)\big) & * Case\ 3 \quad \times \end{cases}$

Since $f(n) = n^{0.75} + n^{-0.5} \in \theta(n^c) = \theta(n^{0.75})$

Then by Case 2, $T(n) \in \theta(n^c \log n) = \theta(n^{0.75} \log n)$

(b) $T(n) = \begin{cases} c & n < 8 \\ 8\,T\left(\frac{n}{8}\right) + n^{\ln n} & n \geq 8 \end{cases}$

By Master's Theorem: $a = 8 \quad b = 8 \quad c = \log_b a = \log_8 8 = 1$

$f(n) = n^{\ln n} \in \begin{cases} O(n^{c-\varepsilon})? \Rightarrow T(n) \in \theta(n^{0.75}) & * Case\ 1 \quad \times \\ \theta(n^c \log^k n)? \Rightarrow T(n) \in \theta(n^{0.75} \log^{k+1} n) & * Case\ 2 \quad \times \\ \Omega\,(n^{c+\varepsilon})? \Rightarrow T(n) \in \theta\big(f(n)\big) & * Case\ 3 \quad \surd \end{cases}$

Case 1: $\lim_{n\to\infty} \frac{n^{\ln n}}{n^{c-\varepsilon}} = n^{\ln n - c + \varepsilon} = n^{\ln n - 1 + \varepsilon} = \infty \quad \times$

Case 2: $\lim_{n\to\infty} \frac{n^{\ln n}}{n^c \log^k n} = \lim_{n\to\infty} \frac{n^{\ln n}}{n^1 \log^k n} = \lim_{n\to\infty} \frac{n^{\ln n - 1}}{\log^k n} = \lim_{n\to\infty} \frac{1 + n^{-1.25}}{\log^k n} = 0 \quad \times$

Case 3: $\lim_{n\to\infty} \frac{n^{\ln n}}{n^{c+\varepsilon}} = n^{\ln n - c - \varepsilon} = n^{\ln n - 1 - \varepsilon} = \infty \quad \surd$

Therefore, the tight bounds for it is is $T(n) \in \theta\big(f(n)\big) = \theta(n^{\ln n})$.

(c) $T(n) = \begin{cases} c & n < 11 \\ 19\,T\left(\frac{n}{11}\right) + n\sqrt[5]{n} & n \geq 11 \end{cases}$

By Master's Theorem: $a = 19 \quad b = 11 \quad c = \log_b a = \log_{11} 19 = 1.228$

$f(n) = n\sqrt[5]{n}$

$= n^{1.2} \in \begin{cases} O(n^{c-\varepsilon})? \Rightarrow T(n) \in \theta(n^{1.228}) & * Case\ 1 \quad \surd \\ \theta(n^c \log^k n)? \Rightarrow T(n) \in \theta(n^{1.228} \log^{k+1} n) & * Case\ 2 \quad \times \\ \Omega\,(n^{c+\varepsilon})? \Rightarrow T(n) \in \theta\big(f(n)\big) & * Case\ 3 \quad \times \end{cases}$

$Case\ 1: \lim_{n\to\infty} \frac{n^{1.2}}{n^{c-\varepsilon}} = n^{1.2-c+\varepsilon} = n^{1.2-1.228+\varepsilon} = 0, for\ any\ \varepsilon\ s.t. 0 < \varepsilon <\approx 0.228.$ √

$Case\ 2: \lim_{n\to\infty} \frac{n^{1.2}}{n^{c}\log^k n} = \lim_{n\to\infty} \frac{n^{1.2}}{n^{1.228}\log^k n} = \lim_{n\to\infty} \frac{1}{n^{0.028}\log^k n} = 0$ ×

$Case\ 3: \lim_{n\to\infty} \frac{n^{1.2}}{n^{c+\varepsilon}} = n^{1.2-c-\varepsilon} = n^{1.2-1.228-\varepsilon}, does\ not\ work\ with\ any\ \varepsilon\ s.t.\ \varepsilon > 0.$ ×

Therefore, the tight bounds for it is is $T(n) \in \theta\big(f(n)\big) = \theta(n^{1.228})$.

$$(d) T(n) = \begin{cases} c & n \le 1 \\ \frac{6}{5}\, T\left(\frac{6n}{7}\right) + n\sqrt[5]{n} & n > 1 \end{cases}$$

By Master's Theorem: $a = \frac{6}{5} \qquad b = \frac{6}{7} \qquad c = \log_b a = \log_{\frac{6}{7}} \frac{6}{5} = 1.183$

$$f(n) = n\sqrt[5]{n}$$

$$= n^{1.2} \in \begin{cases} O(n^{c-\varepsilon})? \Longrightarrow T(n) \in \theta(n^{1.183}) & *\ Case\ 1 \text{ ×} \\ \theta(n^c \log^k n)? \Longrightarrow T(n) \in \theta(n^{1.183}\log^{k+1} n) & *\ Case\ 2 \text{ ×} \\ \Omega\,(n^{c+\varepsilon})? \Longrightarrow T(n) \in \theta\big(f(n)\big) & *\ Case\ 3 \text{ √} \end{cases}$$

$Case\ 1: \lim_{n\to\infty} \frac{n^{1.2}}{n^{c-\varepsilon}} = n^{1.2-c+\varepsilon} = n^{1.2-1.183+\varepsilon} = \infty, for\ any\ \varepsilon\ s.t.\ \varepsilon > 0.$ ×

$Case\ 2: \lim_{n\to\infty} \frac{n^{1.2}}{n^{c}\log^k n} = \lim_{n\to\infty} \frac{n^{1.2}}{n^{1.183}\log^k n} = \lim_{n\to\infty} \frac{n^{0.017}}{\log^k n} = 0$ ×

$Case\ 3: \lim_{n\to\infty} \frac{n^{1.2}}{n^{c+\varepsilon}} = n^{1.2-c-\varepsilon} = n^{1.2-1.183-\varepsilon} = \infty, for\ any\ \varepsilon\ s.t. 0 < \varepsilon <\approx 0.017.$ √

Therefore, the tight bounds for it is is $T(n) \in \theta\big(f(n)\big) = \theta(n^{1.2})$.

$$(e) T(n) = \begin{cases} c & n \le 1 \\ \frac{3}{5} T\left(\frac{3n}{5}\right) + \frac{c}{n} & n > 1 \end{cases}$$

Because $a = \frac{3}{5} \le 1$, *Master's Theorem does not apply.*

We can try algebraic iteration to see if it would yield a result.

$$T(n) = \frac{3}{5}\, T\left(\frac{3}{5}n\right) + \frac{c}{n}$$

$$= \frac{3}{5}\left(\frac{3}{5}\, T\left(\frac{3}{5}\cdot\frac{3}{5}n\right) + \frac{3}{5}\cdot\frac{c}{n}\right) + \frac{c}{n}$$

$$= \left(\frac{3}{5}\right)^2 T\left(\left(\frac{3}{5}\right)^2 n\right) + \frac{2c}{n}$$

Following this pattern, $= \left(\frac{3}{5}\right)^k T\left(\left(\frac{3}{5}\right)^k n\right) + \frac{kc}{n}$ *after* k^{th} *iterations.*

We guess that $T(n) = \left(\frac{3}{5}\right)^k T\left(\left(\frac{3}{5}\right)^k n\right) + \frac{kc}{n}$ *and try to prove it via induction.*

Base case: $\left(\frac{3}{5}\right)^k n = 1$

$$\left(\frac{3}{5}\right)^k = \frac{1}{n}$$

$$k = \frac{\log n}{\log \frac{5}{3}}$$

Plug in k, $T(n) = \frac{1}{n}T(1) + \frac{log\, n}{log\frac{5}{3}} \cdot \frac{c}{n}$

$$= \frac{c}{n} + \frac{c}{log\, \frac{5}{3}} \cdot \frac{log\, n}{n}$$

Then, the runtime for this algorithm is $T(n) = \frac{c}{n} + \frac{c}{log\frac{5}{3}} \cdot \frac{log\, n}{n}$.

Therefore, the tight bounds for it is is $T(n) \in \theta\left(\frac{log\, n}{n}\right)$.

2. *Kleinberg and Tardos: Chapter 2, Exercise 6.*

(a) *For some function f that you should choose, give a bound of the form* $O\big(f(n)\big)$ *on the running time of this algorithm on an input of size n.*

```
From the textbook: 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
```

First, we need to figure out the inner for loop, which include adding up array entries and storing the result. Then, calculate how many times does the inner for loop run within the outer for loop.

In the algorithm that the exercise provided, the step of adding up array entries $A[i]$ *through* $A[j]$ *will take* $j - i$ *steps, and each of the steps takes a constant number* x *of operations. Therefore, the cost of the adding entries is* $\sum_{i+1}^{n} x\,(j - i)$.

Then, storing the result in $B[i, j]$ *will run a constant number* y *of operations.*

Combining the two steps (adding and storing) together, and it will happen $n - i$ *times. Thus, the inner loop will cost* $\sum_{i+1}^{n} x\,(j - i) + y$.

$$\sum_{i+1}^{n} x\,(j - i) + y = \sum_{i+1}^{n} xj - xi + y$$

$$= x\sum_{i+1}^{n} j - x\sum_{i+1}^{n} i + \sum_{i+1}^{n} y$$

Since $\sum_{z}^{n} j = -\frac{(z-n-1)(z+n)}{2}$, *we can substitute* z *with* $i + 1$.

$$= \frac{x(n-i)(i+1+n)}{2} + (n-i)(-xi+y)$$

$$= \frac{(n-i)\left(x(i+1+n)\right) + 2(-xi+y))}{2}$$

$$= \frac{(n-i)\,(-xi+nx+x+2y)}{2}$$

The outer for loop will run the inner loop n times. Thus together, the cost of the algorithm is $T(n) = \sum_1^n \frac{(n-i)\,(-xi+nx+x+2y)}{2}$.

$$\sum_1^n \frac{(n-i)\,(-xi+nx+x+2y)}{2}$$

$$= \sum_1^n \frac{-2nxi + n^2x + nx + 2ny + xi^2 - ix - i2y}{2}$$

$$= \sum_1^n \frac{xi^2 - i(2nx+x+2y) + n(nx+x+2y)}{2}$$

$$= \frac{x}{2}\sum_1^n i^2 - \frac{2nx+x+2y}{2}\sum_1^n i + \sum_1^n \frac{n(nx+x+2y)}{2}$$

Since $\sum_1^n i^2 = \frac{n(n+1)(2n+1)}{6}$ *and* $\sum_1^n i = \frac{n(n+1)}{2}$.

$$= \frac{xn(n+1)(2n+1)}{12} - \frac{(2nx+x+2y)n(n+1)}{4} + \frac{n^2(nx+x+2y)}{2}$$

$$= \frac{xn^3 - nx - 3ny + 3n^2y}{6}$$

$$Thus, T(n) = \frac{x}{6}n^3 + \frac{y}{2}n^2 - \frac{x+3y}{6}$$

Now, we can apply the limit test.

$$\lim_{n\to\infty} \frac{T(n)}{n^3} = \lim_{n\to\infty} \frac{\frac{x}{6}n^3 + \frac{y}{2}n^2 - \frac{x+3y}{6}}{n^3} = \frac{x}{6}$$

Since the limit test yields a positive, non-zero result, then $T(n) \in \theta(n^3)$.

And $T(n) \in \theta(n^3)$ *means both* $T(n) \in O(n^3)$ *and* $T(n) \in \Omega(n^3)$.

Therefore, for part (a), a bound of $O(f(n))$ *for this algorithm is* $T(n) \in O(n^3)$.

(b) *Show that the running time of the algorithm on an input of size n is also* $\Omega\big(f(n)\big)$.

Same as the steps from part (a), a bound of $\Omega(f(n))$ *is* $T(n) \in \Omega(n^3)$.

(c) *Give a different algorithm to solve this problem, with an asymptotically better running time.*

To find a better running time, we need to create an algorithm whose run time is less than n^3*, or fewer for (or while) loops.*

Instead of adding up the array entries A[i] through A[j] in a loop, we could minimize that and combine it with the existing for loop.

We can just fill in the blanks as the for loops iterations of i *and* j *go.*

Pseudo code:

```
finding_sums(list A)
    initialize B as a nested list
    for i in range(n):
        for j in range(n-i):
            j = i+j+1
            if B[i, j-1]:
                B[i, j] = A[i] + A[j]
            else:
                B[i, j] = 0
    return B
```

This way, we lost a for loop and added an if statement, which will only add a constant number of operations.

Thus, the algorithm will yield faster run time, such that $T(n) \in O(n^2)$.

3. *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.*

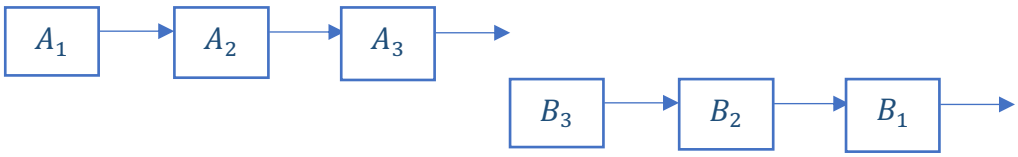
   For simplicity, imagine a scenario that there are only 3 jobs that needs to be worked on for both machines, such that the series of jobs are A_1, A_2, A_3 *and* B_1, B_2, B_3.

Assume the order of A_1, A_2, A_3 to going to the copying machine is by ascending order of their number (e.g. A_1, A_2, A_3).

Then, there are a worst scenario (Case 1) and a best scenario (Case 2) for the machines to work together.

Case 1: Copying machine A_1, A_2, A_3 , and the binding machine B_3, B_2, B_1.

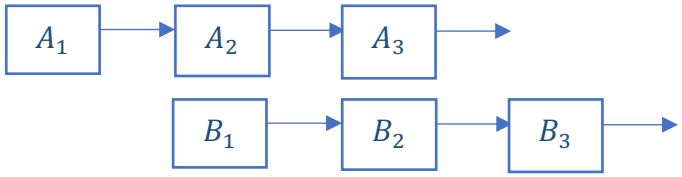
In order for the binding machine to start working on B_3, it has to wait for the copying machine to finish with A_3, since the each of the document only enters the binding machine after it has been copied.



This the worst time scenario, where the binding machine cannot work until the copying machine is finished with all the documents.

This way, the total time cost will be huge comparing to other cases.

Case 2: Copying machine A_1, A_2, A_3 , and the binding machine B_1, B_2, B_3.



In this scenario, B_1 can start immediately after A_1. There will not be any waste of time for the binding machine.

Of course, there are other scenarios where A_i's and B_i's orders are mixed up, but they will not be as horrible as the worst case or as easy as the best case.

Therefore, if the documents are processed by the same order on both of the machines, the binding machine will not be losing time.

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

This question is asking us to find a tight lower bound to the total idle time resulting from sequence S.

I do not know how to answer this question. :(

(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.*

To aim for shortest working time, we can start by a working list that contains all of

the copying and binding jobs. Initialize another empty list for the algorithm to fill up as the output.

Find the minimum working time in the working list and check its job type.

If it is copying, put it (say A_i)and its corresponding binding job (B_i) in front of the output list, with A_i be in front of B_i.

Or if it is B_i, put it in the end of the output list with its corresponding copying job (A_i), with A_i be in front of B_i.

Proceed until the working list is empty and output the resulting list.

Pseudo code:

```
sort_jobs(list A, list B)
    working_list = combining the two lists
    result = 0
    while working_list:
        find the job with the shortest time
        if A:
            append it in front of result with its corresponding B
        else:
            append it in the end of result with its corresponding A
        remove it from working_list
    return result
```

4. *(Bonus) Let* **void** sort2(**int**[] a, **int** i, **int** j) *be a method that puts elements* a[i] *and* a[j] *from an array in ascending order. Express the asymptotic (Big-Theta) running time of the following algorithm as a function of the input length n:*

```
void kme(int[] a) {
    int n = a.length;
    if (n < 2) {
        return;
    }
    int t = 1;
    while (t < n - t) {                          → Loop #1
        t += t;
    }
    for (int p = t; p > 0; p >>= 1) {            → Loop #2
        for (int i = 0; i < n - p; ++i) {        → Loop #3
            if ((i & p) != 0) {
                sort2(a, i, i+p);                → If statement #1
            }
        }
        for (int q = t; q > p; q >>= 1) {        → Loop #4
            for (int i = 0; i < n - q; ++i) {    → Loop #5
                if ((i & p) != 0) {
                    sort2(a, i+p, i+q);          → If statement #2
```

```
                }
            }
        }
    }
}
```

Finding the length of n and checking the base case will take a constant c runtime.

Loop #1 will run $\log n$ times, since t will double itself of each iteration.

Loop #2 will run $\log n$ times, since there will be t iterations in total.

Loop #3 will run $\log n$ times, since there will be $n - t$ iterations in total.

Assume if statement #1 is true on every iteration, and **`sort2`** *only does simple swapping if element $a[j]$ is larger than $a[i]$, then it will take constant run time as well. It will run c times altogether.*

Loop #4 will run $\log n$ times, since there will be t iterations in total.

Loop #5 will run $\log n$ times, since there will be $n - t$ iterations in total.

Assume if statement #2 is true on every iteration in the worst case, and **`sort2`** *only does simple swapping if element $a[j]$ is larger than $a[i]$, then it will take constant run time as well. It will run c times altogether.*

Then, $T(n) = \theta(c + \log n + \log n(\ (\log n \cdot c) + \log n\ (\log n \cdot c)))$

$= \theta(c + \log n + \log n(\ c \log n + \log^2 n))$

$= \theta(c + \log n + c \log^2 n + c \log^3 n)$

$= \theta(\log^3 n)$

Thus, the total runtime for the algorithm is $\theta(\log^3 n)$.