Homework 1

Miguel A. Montoya Z. (A01226045)

August 23, 2017

1. Demonstrations

   (a) Using approximation by integrals we can get both, the upper and lower bound. We need to remember that

   $$\int_{m}^{n+1} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m-1}^{n} f(x)dx$$

   We can start counting from k=2, and add 1

   $$\sum_{k=1}^{n} \frac{1}{k^2} = 1 + \sum_{k=2}^{n} \frac{1}{k^2} \leq 1 + \int_{1}^{n} \frac{dx}{x^2} = 1 - \frac{1}{n} + 1 = 2 - \frac{1}{n} \leq 2$$

   $$\sum_{k=1}^{n} \frac{1}{k^2} \leq 2 - \frac{1}{n} \leq 2$$

   (b) Using linearity

   $$\sum_{k=1}^{n} \mathcal{O}(f_k(i)) = \sum_{k=1}^{n} c_k(f_k(i)) = c_1(f_k(1)) + c_2(f_k(2)) + \ldots + c_n(f_k(n))$$

   If we take $c_{max}\ \{1 \leq k \leq n\}$ then $\sum_{k=1}^{n} c_k(f_k(i)) \leq c_{max} \left(\sum_{k=1}^{n} f_k(i) \right) = \mathcal{O}\left(\sum_{k=1}^{n} f_k(i) \right)$

   (c) When n = 1

   $$1^2 = \frac{1(1+1)(2*1+1)}{6} = 1$$

   Assumme its true for k

   $$1^2 + 1^2 + \ldots + k^2 = \frac{k(1+k)(2k+1)}{6}$$

   Prove for k+1

   $$1^2 + 1^2 + \ldots + k^2 + (k+1)^2 = \frac{(k+1)(1+(k+1))(2(k+1)+1)}{6}$$

$$1^2 + 1^2 + \ldots + k^2 + (k+1)^2 = \frac{(k+1)(k+2)(2k+3)}{6}$$
$$\frac{k(1+k)(2k+1)}{6} + (k+1)^2 = \frac{(k+1)(k+2)(2k+3)}{6}$$
$$\frac{k(1+k)(2k+1)}{6} + \frac{6(k+1)^2}{6} = \frac{(k+1)(k+2)(2k+3)}{6}$$
$$k(1+k)(2k+1) + 6(k+1)^2 = (k+1)(k+2)(2k+3)$$
$$(2x^3 + 3x^2 + x) + (6x^2 + 12x + 6) = (2x^3 + 9x^2 + 13x + 6)$$
$$2x^3 + 9x^2 + 13x + 6 = 2x^3 + 9x^2 + 13x + 6$$
$$1 = 1$$

Thus, k+1 is true

(d) Recursive solution of plane division

$$P_{n+1} = P_n + (n+1)$$
$$P_{n+1} = P_{n-1} + [(n) + (n+1)]$$
$$P_{n+1} = P_{n-2} + [(n-1) + (n) + (n-1)]$$
$$\vdots$$
$$P_{n+1} = P_2 + [3 + \ldots + P_{n-1} + (n) + (n-1)]$$
$$P_{n+1} = P_1 + [2 + 3 + \ldots + P_{n-1} + (n) + (n-1)]$$
$$P_{n+1} = P_0 + [1 + 2 + 3 + \ldots + P_{n-1} + (n) + (n-1)]$$
$$P_{n+1} = 1 + [1 + 2 + 3 + \ldots + P_{n-1} + (n) + (n-1)] = 1 + \frac{(n+1)(n+2)}{2}$$

Thus
$$P_n = 1 + \frac{n(n+1)}{2} = \frac{n^2 + n + 2}{2}$$

2. Comb sort

(a) Count i = Number of times the gap is going to be decreased.

def combsort(A):
gap = len(A) c_1
sorted = False c_2
shrinkFactor = 1.3 c_3
if gap < 2: c_4
    returnc_5
while !sorted: $c_6(i+1)$
    gap = floor(int(gap/shrinkFactor)) $c_7 i$
    if gap > 1$c_8 i$
        sorted = False $c_9 i$
    else $c_{10} i$
        gap = 1 $c_{11} i$
        sorted = True $c_{12} i$
    k = 0 $c_{13} i$
    while k + gap < len(A): $c_{14} \sum_{j=1}^{i}(N - \lfloor \frac{N}{1.3^i} \rfloor) + 1$
        if A[i] > A[i + gap]: $c_{15} \sum_{j=1}^{i}(N - \lfloor \frac{N}{1.3^i} \rfloor)$
            A[i], A[i + gap] = A[i + gap], A[i] $c_{16} \sum_{j=1}^{i}(N - \lfloor \frac{N}{1.3^i} \rfloor)$
            sorted = False $c_{17} \sum_{j=1}^{i}(N - \lfloor \frac{N}{1.3^i} \rfloor)$
        k += 1 $c_{18} \sum_{j=1}^{i}(N - \lfloor \frac{N}{1.3^i} \rfloor)$

(b) Cases

i. Worst case: Numbers are sorted, in reverse. It will make the comparison and swap for every number $\mathcal{O}(n^2)$

ii. Average case: Numbers are randomized first, then processed.

iii. Best case: Numbers are ordered. It never makes the comparison nor the swap. $\Theta(n\,log(n))$

(c) Correctness

The loop invariance for this algorithm is: for every element in$i, i{+}gap$ should be $> i$

3. Book's exercises

(a) Cormen

i. Use merge sort (Which is $\Theta(n\,lg(n))$ and the following algorithm: This algorithm runs at most $\mathcal{O}(n)$, given that the worst cas scenario is to go trough every element, decreasing n by 1 at each step. This leaves the time complexity to the merge sort.

```
Use Merge Sort to sort array in Θ(n lg(n))
leftP = 0
rightP = n-1
while i < j:
    if A[i] + A[j] = S: //If the elements have been found
        return True
    if A[i] + A[j] < S: //If smaller, increase (Move right leftP)
        i++
    if A[i] + A[j] > S: //If bigger, decrease (Move left rightP)
        j—
return False
```

ii. $2^{n+1} = \mathcal{O}(2^n)$ and $2^{2n} = \mathcal{O}(2^n)$

$$2^{n+1} = 2^n * 2 \leq c * 2^n = \mathcal{O}(2^n).\text{ So } 2^{n+1} = \mathcal{O}(2^n) \text{ if } 2 \leq c$$

$$2^{2n} = 2^n{*}2^n \leq c{*}2^n = \mathcal{O}(2^n).\text{ For that to be true } 2^n \leq c \text{ but } c \text{ is a constant}$$

(b) Dasgupta

$$\text{Sum of nodes at each level: } 1 + d + d^2 + d^3 + \ldots d^h = \sum \frac{d^{h+1} - 1}{d - 1}$$

$$\text{The nodes of the d-tree, when counting to } d^{h+1} = n(d-1) + 1$$

$$\text{Then } d^{h+1} \geq n + 1$$

$$log(d^{h+1}) \geq log(n+1)$$

$$(h+1)log(d) \geq log(n+1)$$

$$(h+1)log(d) \geq log(n)$$

$$h \geq \frac{log(n)}{log(d)} \to h = \Omega\left(\frac{log(n)}{log(d)}\right) \text{ Thus, proving the depth}$$

$$h \geq log_d(n) \text{ This is the minimum depth}$$

4. Asymptotic bounds

(a) $T(n) = \begin{cases} T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\frac{2n}{4}\right) & \text{if } n \geq 4 \\ 4 & \text{if } n < 4 \end{cases}$

$$\text{Note: } \frac{2n}{4} = \frac{n}{2} \text{ and assume } n = 2^b$$

$$\text{Then } 2T\left(\frac{n}{2}\right) \text{ if } n \geq 4$$

$$\text{Guess: } T(n) = \mathcal{O}(n\, log(n))$$

$$\text{Bound holds because we assume } \frac{n}{2} \leq n$$

$$T(\frac{n}{2}) \leq c\frac{n}{2}\, lg(\frac{n}{2})$$

$$\text{Then } T(n) \leq 2(c\frac{n}{2}\, lg(\frac{n}{2}))$$

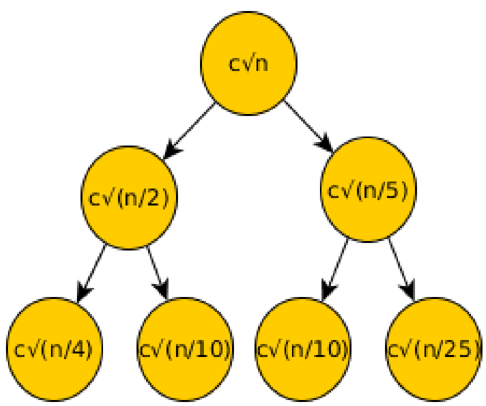
$$T(n) \leq cn\, lg(\frac{n}{2})$$

$$T(n) \leq cn\ (lg(n) - lg(2))$$

$$T(n) \leq cn\, lg(n)$$

$$T(n) = \mathcal{O}(n\, lg(n))$$

(b) $T(n) = \begin{cases} T\left(\frac{n}{2}\right) + T\left(\frac{n}{5}\right) + \sqrt{n} & \text{if } n \geq 4 \\ 4 & \text{if } n < 4 \end{cases}$ Because the properties of



the Tree Method, we can see that the biggest number will be $c\sqrt{n}$. So we can try to probe that the time complexity will be $\mathcal{O}(\sqrt{n}lg(n))$.

(c) $T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + g(n) & \text{if } n \geq 2 \\ 1 & \text{if } n = 1 \end{cases}$ is T(n)=O(n) if g(n)=o(n)

Following the second case of the master method: $a = 2, b = 2$ and $d = 1$ (Because: $o(n) = o(n^1)$)

$$T(n) = \mathcal{O}(n^1 log(n))$$

So no, T(n) is not equal to O(n).

(Also, because the nature of little oh, it always should be bigger than the function. A regular O can be equal)