תרגיל בית 4 :מבני נתונים ואלגוריתמים (094224)חורף תשפ"ב

שאלה 1:

.1

```
MIN_MAX(T)
        new array keys [1 … n]
        inorder(T, T.root, keys)
        max = n
        min = 1
        while (max > min)
                print(keys[max])
                print(keys[min])
                max--
                min++

index = 1
inorder(T, v, keys):
        if(index == n OR v == NIL)
                return
        if (v == leaf)
                keys[index] = v.key
                index++

        inorder(T, v.left, keys)

        keys[index] = v
        index++

        inorder(T, v.right, keys)
```

נקרא לפונקציה inorder ונשלח לה את העץ השורש ומערך ריק שהיא אמורה להחזיר את המערך עם המפתחות של האובייקטים לפי סדר inorder שלהם שזה המפתחות בסדר ממוין לא יורד ואז מדפיסים את התא האחרונה במערך ואחר כך התא הראשונה ואז התא השנייה מבסוף ואז התא השנייה… עד המצביע הימני והשמאלי מחליפים סדר

למה: בשלב k של הפעולה inorder המערך keys יכיל את k המפתחות הקטנות ביותר בסדר לא יורד

נוכיח באינדוקציה:

בסיס: עבור k = 1 התנאי מתקיים באופן טריוויאלי (מהגדרת עץ חיפוש בינארי הצומת הכי שמאלית היא המינימלית ומהגרת ה inorder בשלב 1 אנו לוקחים את הצומת הזאת)

הנחה: נניח נכונות עבור k < n

צעד: עבור k = n מהגדרת עץ חיפוש בינארי ומהפעולה inorder בשלב זה נוסיף את הצומת הימני ביותר שהוא גם המקסימלי מבין המפתחות ולכך המערך נשאר ממוין בסדר לא יורד

כלומר הוכחנו שהפעלת inorder על עץ חיפוש בינארי מחזירה מערך ממוין בסדר לא יורד של המפתחות

ב.

we modify the heap and add an attribute:

value - holds a number

```
K_Nearest(A, x, k)
    new max heap H
    for i = 0 to k do
        H.insert(|A[i] - x|)
        H.value = A[i]

    for i = k+1 to n do
        if |A[i] - x| <= |H[1] - x|
            Heap_Extract_Max(H)
            Heap_insert(H, A[i])
```

Run-time:

the first for is O(k)

the second for is :

Heap_Extract_Max and Heap_insert both are O(log(k)) (from lectures)

each of those lines are executed at most O(n - k + 1)

therefor the runtime of the for loop is O((n-k+1) * log(k)) = O(nlog(k))

since n > k

Correctness:

the key in the root of the heap is the maximum distance of the k numbers in the heap and x and if we find a number in the k+1, n numbers in the array that's distance from x is less than the maximum we replace those numbers

שאלה 2 :

נסתכל על מבני הנתונים הבא: (עץ 2-3)

(we use all of the functions that are used in this pseudocode in the lectures, except for those that are redefined in the next lines)

all attributes are the same as 2-3 tree

with the additional Node attributes:

min = minimum key in it's subtree

n = number of leaves in it's subtree (the users data)

sum = the sum of the key's in it's subtree

Init(D)

```
2_3_init(D):
D.root.n = 0
D.root.sum = 0
D.root.min = −∞
```

Run-time = O(1)

2_3_init: O(1)

the rest of the lines are O(1)

let's override the function Update_key(x) that we defined in the lectures to

Update_key(x)

```
x.key = x.left.key
x.min = x.left.min
x.n = x.left.n
x.sum = x.left.sum

if x.middle ≠ NIL then
    x.key = x.middle.key
    x.n += x.middle.n
    x.sum += x.middle.sum

if x.right ≠ NIL then
    x.key = x.right.key
    x.n += x.right.n
    x.sum += x.right.sum
```

Run-time: O(1)

Insert(D, x)

```
2_3_Insert(D, x)
```

Delete(D, x)

```
2_3_Delete(D, x)
```

Run-time: we've seen in the tutorials that the runtime of insert, delete is O(log(n))

helper(v, k)

```
sum = n = 0
if v.min > k
    return sum, n

while v is not leaf
```

```
            if v.left.key >= k
                v = v.left
            else if v.middle.key >= k
                sum += v.left.sum
                n += v.left.n

                v = v.middle

            else if v.right is not NIL and v.right.key >= k
                sum += v.left.sum + v.middle.sum
                n += v.left.n + v.middle.n

            else
                break

        if v.key == ∞
            print 'Error empty tree'

        if v.key <= k
            sum += v.sum
            n += v.n
        return sum, n
```

AverageOf($D,\ k_1,\ k_2$)

```
        if k2 < k1
            return 0
        tup1, tup2 = helper(D.root, k2), helper(D.root, k1)
        return (tup1[1] - tup2[1]) / (tup1[2] - tup2[2])
```

Run-time:

אנו רואים כי לולאת ה while בפונקציה helper מבצעת במקרה הכי גרוע $O(log(n))$ ריצה בהצומת v עד לעלה כלשהו

שאלה 3: נשתמש בפעולות שמצאנו בתרגול 9:

1-Build Heap(A) – לייצר ערימה חדשה בגודל של k+1
2- Heap Insert T(A, H.root)
3-For(i=1;i<=k;i++) עכשיו נריץ לולאה המתבצעת k פעמים
3.1-x=Heap Extract Min(A)
3.2-print(x)
3.3-Heap Insert(A, x.right)
3.4-Heap Insert(A, x.left)

הסבר:
מהגדרת ערימת מינימום אנו יודעים שהאיבר בשורש הוא המינימלי ביותר ולכן הוא האיבר הראשון שנדפיס.
יצרנו ערימה חדשה כדי לשמור על הערימה המקורית ,בנוסף לעולם לא נשמור יותר מ k+1 איברים (מכיוון שבכל פעם אנו מחזירים איבר אחד ומוסיפים 2 איברים (הבנים שלו הימני והשמאלי) k פעמים
1+k-2k=k+1
ולכן גודל הערימה הוא לכל היותר k+1.
בכל איטרציה נוציא האיבר המינימלי מבין כל הנמצאים בערימה ואז נשים הבנים שלו שהם הקטנים אחריו שנשארו בערימה המקורית כי כבר האח שלו נמצא בערימה כי אנחנו שמים אותם באותו איטרציה.
זמן ריצה:

1-O(k+1)=O(k)
2-O(log(k+1))=O(log(k))
3-O(k)
3.1-O(log(k+1))=O(log(k))
3.2-O(1)
3.3+3.4-O(log(k+1))=O(log(k)

ולכן בסה"כ:

O(k)+O(log(k))+O(k*log(k))=O(k*log(k))

כפי שנדרש.

שאלה 4:

א- קיים C_1 כך ש:

$\forall n>1: T(n) \leqq T(n/5)+T(3n/4)+C_1 n$
$n=1: \quad T(n)=C_1$

טענה: עבור $C_1 \leqq C_2$:

$T(n) \leqq C_2 * n$

נוכיח באינדוקציה :
בסיס : עבור n=1 :

$T(1)=C_1 \leqq C_2$

נניח עבור כל k<n שמתקיים : $T(k) \leqq C_2 * k$
ונוכיח עבור n :

$T(n) \leqq T(n/5)+T(3*n/4)+C_1 n \leqq C_2 * n/5 + C_2 * 3*n/4 + C_1 n \leqq C_2 * n/5 + C_2 * 3*n/4 + C_2 n = C_2 * [n/5+3*n/4+n] = C_2 * 1.95*n$

ומכאן נסיק ש $T(n) \leqq O(n)$

ב- קיים C כך ש:

$\forall n>1: T(n)=2*T(n/4)+Cn^2$

$n=1: \quad T(n)=C$

טענה :

$T(n) \leqq C*n^2$

נוכיח באינדוקציה :

בסיס : עבור $n=1$:

$T(1)=C*1^2$

נניח עבור כל $k<n$:

$T(k) \leqq C*k^2$

נוכיח עבור n:

$T(n)=2*T(n/4)+Cn^2 \leqq 2*C*(n/4)^2+Cn^2=Cn^2*[2/16+1]=9/8*Cn^2$

ומכאן נסיק ש $T(n) \leqq O(n^2)$

ג- קיים C כך ש:

$\forall n>1: T(n)=T(n-1)+C*2^n$

$n=1: \quad T(n)=C$

$T(n)=T(n-1)+C*2^n=T(n-2)+C*2^{n-1}+C*2^n=T(n-3)+C*2^{n-2}+C*2^{n-1}+C*2^n=....=$

$T(2)+C*2^3+.....+C*2^{n-2}+C*2^{n-1}+C*2^n=T(1)+C*2^2+C*2^3+.....+C*2^{n-2}+C*2^{n-1}+C*2^n=$

$C+C*2^2+C*2^3+.....+C*2^{n-2}+C*2^{n-1}+C*2^n=C+C*[2^2+2^3+.....+2^{n-2}+2^{n-1}+2^n]=$

$C+C[2^2(2^n-1)]=C*[1+2^{n+2}-4]=C[2^{n+2}-3]=4C2^n-3C$

ולכן $T(n)=\boldsymbol{\Theta}(2^n)$ ובמיוחד $T(n) \leqq O(2^n)$.

שאלה 5:

א- מהרצאה למדנו שללא קשר לערך ה, ח-החציון נמצא ב סטטיסטי הסדר $\left\lfloor\frac{n+1}{2}\right\rfloor$ ולכן מהגדרת הסכומים:

$$\sum_{x_i<x_{\left\lfloor\frac{n+1}{2}\right\rfloor}} w_i = \sum_{i=1}^{\left\lfloor\frac{n+1}{2}\right\rfloor-1} w_i = \sum_{i=1}^{\left\lfloor\frac{n+1}{2}\right\rfloor-1} \frac{1}{n} = (\left\lfloor\frac{n+1}{2}\right\rfloor - 1) * \frac{1}{n} \leq (\frac{n+1}{2} - 1) * \frac{1}{n} = \frac{n-1}{2n} < \frac{n}{2n}$$
$$= \frac{1}{2}$$

$$\sum_{x_i>x_{\left\lfloor\frac{n+1}{2}\right\rfloor}} w_i = \sum_{\left\lfloor\frac{n+1}{2}\right\rfloor+1}^{n} w_i = \sum_{\left\lfloor\frac{n+1}{2}\right\rfloor+1}^{n} \frac{1}{n} = \frac{1}{n} * [n - \left\lfloor\frac{n+1}{2}\right\rfloor] \leq \frac{1}{n} * [n - (\frac{n+1}{2}) - 1]$$
$$= \frac{1}{n} * \frac{2n-n-1-2}{2} = \frac{1}{n} * \frac{n-3}{2} = \frac{n-3}{2n} = \frac{1}{2} - \frac{3}{2n} \leq \frac{1}{2}$$

ב- נגדיר **מבנה נתונים דומה מאוד למערך :**
הוא מערך בעצמו אך לכל איבר מוגדר שתי שדות שהן : (האיברים והמשקליים שלהם)
אפשר להסתכל על המבנה כשני מערכים אך נעדכן אותם בהתאמה

x_1	x_2	x_3							x_n
w_1	w_2	w_3							w_n

1- לשים את כל האיברי עם משקלם הנתון במבנה הנתונים החדש A[1,…,n] .
2- לסווג את מבנה הנתונים לפי ערך האיברים באמצעות Merge Sort(A, 1, n)
3- ועכשווי נרוץ על המערך הממוין כך שנסכם את משקלי האיברים עד לקבלת סכום שווה ל 1/2
4- נחזיר האיבר שנתן לנו סכום שווה ל 1/2 כי הוא החציון שלנו -וזה נובע מיידית מלמה שהוכחנו

מבחינת סיבוכיות :
שורה 1 לוקחת O(n) סיבוכיות
שורה 2 לוקחת O(nlogn)
שורה 3 לוקחת O(n)
שורה 4 לוקחת O(1)
ולכן סה"כ O(nlogn)

ג- נבנה פונקציה שהיא דומה ל Select_Given_Median שראינו בהרצאה מבחינת סיבוכיות ורעיון

```
Weighted_ Median(A, W1  , W2  ){
If n==1 then
  Return A[1]
X=median(A)
k=partition(A,1,n,x)
S1=W1+Sum_Weights (A[1,….,k-1])
S2=W2+ Sum_Weights (A[k+1,…,n])
 If S1<1/2 &&S2≤ 1/2 then
                Return A[k]
Else if S1≥ 1/2 then
Weighted_ Median (A,S1 ,S2+X.w)
Else
Weighted_ Median (A,S1+X.w ,S2)
}
```

בעצם אנחנו שנינו הפונקציה רק מבחינת תנאים :

אחרי שמצאנו החציון בדקנו אם סכום משקלי האיברים הקטנים ממנו הוא קטן מ 1/2 וסכום משקלי האיברים הגדולים ממנו הוא קטן שווה מ 1/2 אז החציון הממושקל הוא החציון הסטטיסטי שיודעים

אחרת אחד מהתנאים לא מתקיים (אחד בדיוק מכיוון שסכום משקלי האיברים ביחד הוא 1) אם סכום משקלי האיברים הקטנים ממנו הוא גדול שווה מ 1/2 אז החציון הממושקל נכלל בצד השמאלי של המערך (האיברים הקטנים ממש מחציון) ולכן נריץ האלגוריתם עוד פעם אך על החלק הנ"ל ונכיל משקל החציון בסכום החלק הימני.

המקרה השני סכום משקלי האיברים הגדולים ממש מהחציון הוא גדול ממש מ 1/2 אז החציון נכלל בצד הימני של המערך (האיברים הגדולים ממש מחציון) ולכן נריץ האלגוריתם עוד פעם אך על החלק הנ"ל ונכיל משקל החציון בסכום החלק השמאלי.

זמן הריצה :

בעצם אנחנו לא שנינו את הפונקציה Select_Given_Median ולכן סיבוכיות הזמן נשארת O(n) מחוץ ל:

-מציאת סכום משקלי האיברים באמצעות הפונקציה Sum_Weights שעוברת על מערך וסוכמת משקלי האיברים ולכן דורש סיבוכיות O(n)