תור עדיפויות

אחד האופנים הנפוצים לשמירת נתונים בזיכרון הוא : **שמירה בצורה ממוינת.** לדרך זו מעלות וחסרונות.

מעלות: חיפוש איבר על פי ערכו יתבצע במהירות על ידי חיפוש בינארי בסיבוכיות O(log N)

חסרונות: הכנסת איבר לאוסף ממוין יתבצע בסיבוכיות O(n) (או O(log N) במקרה של עץ AVL)

לעומת O(1) בהכנסת איבר לאוסף לא ממוין.

מיון אוסף קיים לאחר ההכנסות גם כן יהיה בעלות של לפחות N log N.

לסיכום : המעלה של החיפוש המהיר 'עולה' במחיר של הכנסה איטית

תור עדיפויות הוא מבנה נתונים המהווה מעין פשרה בין אוסף ממוין לאוסף לא ממוין. העדיפות היא מפתח המיון – הערך על פי נרצה למיין את האוסף. אולם לא נחזיק את הנתונים בצורה ממוינת לגמרי אלא באופן שיאפשר גישה לאיבר המקסימלי בקלות ויחד עם זאת הכנסת איבר לאוסף בסיבוכיות נמוכה.

<u>פעולות נחוצות</u>:

חתימת הפונקציה	**מטרה**
Insert(x)	הכנסת ערך לתור העדיפויות.
GetMax()	החזרת האלמנט בעל רמת העדיפות הגבוהה ביותר.
ExtractMax()	הוצאת האלמנט בעל רמת העדיפות הגבוהה ביותר.
Remove(ptr)	הסרת אלמנט המוצבע ע"י ptr.
ChangePriority(ptr,x)	שינוי רמת העדיפות של אלמנט כלשהו הנמצא באוסף.

כיוון שהמבנה משמש להחזקת אלמנטים שהמידע הנחוץ לגביהם זו רמת העדיפות שלהם, חשבו להחזיק את האוסף באופן כזה שהגישה המהירה תהיה אל האלמנט בעל רמת הדחיפות (עדיפות) הגבוהה ביותר, וגם לאחר שאלמנט זה יצא תהיה בידינו הגישה אל האלמנט הבא בעל רמת העדיפות הגבוהה ביותר ברגע זה.

בס"ד

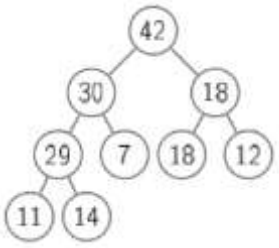
נבחן את סיבוכיות זמן הריצה באפשרויות השונות להחזקת תור עדיפויות:

הפונקציה	מימוש על ידי:			
	מערך (לא ממויין)	**רשימה מקושרת לא ממויינת**	**מערך ממויין**	**רשימה מקושרת ממויינת (דו כיוונית או לחילופין, ערך מקסימלי נמצא בראש)**
Insert(x)	O(1)	O(1)	מציאת המיקום להכנסה מתבצע אומנם ב(O(logn, אבל הזזת האיברים תעלה O(n) .	אין אפשרות לבצע חיפוש בינארי כמו במערך ממויין ולכן: O(n)
GetMax()	O(n)	O(n)	O(1)	O(1)
ExtractMax()	O(n)	O(n)	O(1)	O(1)
Remove(ptr)	O(n)	O(n)	כמו בהכנסה, לעיל.	כנ"ל (כמו בהוספה)
ChangePriority (ptr,x)	O(n)	O(n)	O(n)	כנ"ל..(כבהוספה והסרה)

מימוש יעיל יותר לתור עדיפויות יהיה על ידי **ערימת מקסימום (או להיפך- מינימום)**

הגדרה: ערמה **heap** היא מערך חד ממדי המייצג עץ בינרי שלם או כמעט שלם כאשר כל צומת בעץ גדול או שווה לכל אחד מבניו (בנים, לא צאצאים).

לדוגמא:



ייצוג במערך:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

בנוסף יש לשמור במשתנה **heap_size** את גודל הערמה ובמשתנה **max_size** את מספר האיברים המקסימלי (גודל המערך).

בס"ד

כיצד תתבצענה הפונקציות?

הכנסת ערך לערימה

הערך יתווסף כעלה לתחתית העץ. לאחר מכן, הצומת החדשה תיבדק מול ההורה שלה. אם ההורה מכיל ערך **הקטן** מהצומת החדשה, תתבצע החלפת ערכים ביניהם. כך נמשיך לבדוק את הצומת מול ההורה עד שימצא כתקין (כלומר- ההורה לא קטן מהצומת) או עד ההגעה לשורש העץ.

פעולת תיקון זו נקראת **SiftUp**, תיקון כלפי מעלה.

פעולה הפוכה לזו היא **SiftDown**, המשמשת אותנו בעת הוצאת ערך מקסימלי מהערימה.

הסרת המקסימום מהערימה

כפי שאמרנו, הערך המקסימלי נמצא בשורש העץ. בכדי להסיר אותו מבלי לפגוע בתכונת הערימה נפעל כך: תחילה נחליף את ערך השורש בערך של העלה האחרון(הכי ימני ברמה התחתונה) בעץ. לאחר מכן, "נדרדר" את הערך כלפי מטה כל עוד הערך קטן מערכי הבנים שלו (או מאחד מהם). בכל פעם שנתקל במקרה בו הערך קטן מלפחות אחד הבנים שלו, נחליף אותו עם הבן **הגדול** יותר מבין שני הבנים.

עלות פעולות תיקון אלו תלויות בגובה העץ – מספר הצמתים שיש לסרוק מהעלה ועד השורש, או להיפך. כלומר, העלות היא O(h).

לכן, נרצה לשמר את העץ מה שפחות עמוק (או- כמה שיותר שטוח..). בשביל זה נדאג למלא את כל הרמות הקיימות לפני שניגש "לפתוח" רמה חדשה בעץ. במילים אחרות, נממש את הערימה כעץ בינארי שלם או כמעט שלם. בעץ שלם מספר העלים שווה למספר הצמתים הפנימיים בעץ ועוד 1. מה שגורם לכך שאורך המסלול משורש העץ ועד העלים הינו logn. כלומר, עלות פעולות התיקון תהיה O(logn).

(תזכורת חשובה: עץ שלם או כמעט שלם מתאים מאד למימוש ע"י מערך. ערכי הבנים של צומת i כלשהיא ימצאו באינדקס i*2 ובאינדקס i*2+1 , כאשר שורש העץ נמצא באינדקס 1)

פונקציות ממשק ערימה:

חתימת הפונקציה	הסבר	סיבוכיות
SiftUp(i)	ביצוע תיקון כלפי מעלה, החל מצומת i.	O(logn)
SiftDown(i)	ביצוע תיקון כלפי מטה, החל מצומת i.	O(logn)
Insert(x)	הוספת ערך כעלה ותיקון כלפי מעלה.	O(logn)
GetMax()	החזרת הערך שנמצא בשורש העץ.	O(1)
ExtractMax()	הסרת הערך המקסימלי משורש העץ. (החלפה עם עלה וביצוע תיקון כלפי מטה)	O(logn)
Remove(i)	הסרת אלמנט אידקס i. (כמו הסרת מקסימום, רק עבור צומת כשלהיא בעץ, ולא שורש..בשלב ראשון נבצע שינוי רמת עדיפויות למקסימום ובכך הערך יגיע לשורש העץ, ואז נבצע הסרת מקסימום כרגיל.)	O(logn)
ChangePriority(i,x)	שינוי רמת העדיפות של צומת בעץ.	O(logn)

מימוש הפונקציות:

```
int father(int i)
{
        return (i-1)/2;
}
int leftChild(int i)
{
        return 2*i+1;
}
int rightChild(int i)
{
        return 2*i+2;
}
```

תיקונים – כלפי מעלה וכלפי מטה

הפונקציה הבאה עובדת על ערמה תקינה ומקבלת אינדקס של איבר שיתכן שמפר את תכונת הערמה - קטן מאחד מבניו ומתקנת אותו לפי הצורך – כלפי מטה

```
public void sift_down(int index)
    {
      int l = leftChilde(index), r = rightChilde(index), max = index;
      if (l < heap_size && arr[l] > arr[index])
        max = l;
      if (r < heap_size && arr[r] > arr[max])
        max = r;
      while(max!=index)
      {
          replace(max, index);
        l = leftChilde(index);
         r = rightChilde(index);
        max = index
         if (l < heap_size && arr[l] > arr[index])
                    max = l;
         if (r < heap_size && arr[r] > arr[max])
                max = r;
        }
        }
```

הפונקציה הבאה עובדת על ערמה תקינה ומקבלת אינדקס של איבר שיתכן שמפר את תכונת הערמה - גדול מאביו הפונקציה מתקנת אותו על פי הצורך – כלפי מעלה.

```
public void sift_up(int index)
    {
      while (index >= 0 && arr[index] > arr[parent(index)])
      {
        replace(index, parent_place(index));
        index = parent_place(index);
      }
    }
```

הוספת איבר לערמה כך שתשמר תכונת הערמה

נוסיף את הערך לסוף הערמה – נוסיף עלה ונתקן כלפי מעלה.

```
public void insert(int value)
    {
        if (heap_size==max_size)
                ERORR;
        heap_size= heap_size+1;
        arr[heap_size] = value;
        sift _up(heap_size);

    }
```

החזרת הערך המקסימלי

```
int max()
{
        return arr[1];
}
```

מחיקת מקסימום

נחליף את המקסימום (איבר 1) עם העלה האחרון ונתקן כלפי מטה

```
int extract_max()
{
        int result= arr[1];
        arr[1]=arr[heap_size];
        size=size-1;
        sift_down(1);
        return result;
}
```

מחיקת האיבר הנמצא במקום i

נשנה את ערך האיבר לאין סוף , נתקן כלפי מעלה ונבצע מחיקת מקסימום

```
void remove(int i)
{
	arr[i]=∞;
	sift_up(i);
	extract_max();
}
```

שינוי ערך האיבר הנמצא במקום i (המוצבע על ידי i) ל k

נשנה את ערך האיבר, נשווה ערך ישן לחדש, אם ערך ישן גדול יותר – נתקן כלפי מטה אחרת נתקן כלפי מעלה

```
void change_pariority(int i, int k)
{
	int old=arr[i];
	arr[i]=k;
	if(old>k)
		sift_down(i);
	else
		sift_up(i);
}
```

בניית ערימה עבור n ערכים:

- עלות הכנסת ערך בודד לערימה היא O(logn), בעקבות העלות של פעולת התיקון SiftUp.

ולכן, אם אנו מבצעים n הכנסות שלאחר כל אחת מהן תיקון, הרי שהעלות לבניית הערימה כולה הינה O(nlogn).

- אם אנו מתייחסים למערך קיים, ובו n איברים, ומעוניינים להפוך אותו לערימה תקינה, הרי שניתן לבצע את הפעולה בצורה הפוכה. כך:

נחליט ש- n/2 איברים הנמצאים בימין המערך הינם עלים. כל עלה יכול בהחלט להיחשב כערימת מקסימום תקינה. כעת, ניקח את ההורים של אותם עלים, כלומר כ- n/4 מאברי המערך, ו"נתקן" אותם מול הצאצאים שלהם. איברים אלו נבדקים מול רמה אחת בלבד.

וכן הלאה, n/8 יבדקו מול 2 רמות שתחתם...עד השורש, שהוא האיבר היחיד שעבורו תתבצע פעולת תיקון כלפי מטה ב- O(logn). אם נסכום את העלויות של כלל התיקונים שבוצעו נקבל סכום סדרה הנדסית, כלומר העלות הכוללת תהיה 2n. ז"א, סיבוכיות שמן הריצה היא: O(n) בלבד.

כך תמומש הפונקציה הזו:

```
public void create()
    {
        for (int i = (heap_size-1) / 2; i >= 0; i--)
            sift_down(i);
    }
```

מיון ערימה

מיון זה הוא אחד היעילים והזולים מבחינת זמן ביצוע.

סיבוכיות זמן הריצה הינה: O(nlogn).

כיצד?

תחילה נבצע בניית ערימה עבור n האיברים שאותם יש למיין. O(n).

לאחר מכן, נבצע n פעמים הסרת ערך מקסימלי מערימת מקסימום. שלב זה עולה O(nlogn) בעקבות פעולת התיקון שנדרשת אחרי כל הוצאה.

מימוש:

```
public void Sort()
    {
        for (int i = heap_size; i >= 1; i--)
            arr[i]=extract_max();
    }
```

סה"כ O(nlogn+n) . ה-n זניח, ולכן הסיבוכיות היא O(nlogn).

יתרונות הערמה:

- סיבוכיות: כל הפעולות בסיבוכיות O(log N) חוץ מהחזרת מקסימום – O(1), בנייה- O(n)
- אין צורך לשמור הפניות בין האב לבניו ולהיפך – חיסכון בזיכרון
- כל הפונקציות ממומשות במספר קטן של שורות קוד.

בס"ד

תרגילים

תרגיל 1

נתונים מערכים, קבעי עבור כל מערך אם הוא מיצג ערמת מקסימום, אם לא – נמקי מיהו האיבר המפר את תכונת הערמה

א.

		9	12	7	8	50	40	10	15	60	20	100

ב.

2	16	11	3	14	18	22	40	70	80	50	120	200

ג.

	16	11	3	10	20	15	9	30	45	44	40	50

תרגיל 2

נתונה ערמת מקסימום,

	51	40	30	20	10	64	73	50	45	80	90	100

ביצד תראה הערמה לאחר הוספת המספר 85?

ביצד תראה הערמה לאחר שינוי ערך המספר 90 ל 28?

ביצד תיראה הערמה לאחר מחיקת המקסימום?

תרגיל 3

כתבי פונקציה המקבלת עץ בינרי כמעט שלם ומחזירה את המערך המייצג את העץ בסדר תוכי

תרגיל 4

כתבי פונקציה שמקבלת עץ בינרי ומחזיר True אם העץ הוא ערמה, ו-False אחרת.

כלומר: תחזיר True אם ורק אם העץ הבינרי הוא כמעט שלם והערך בכל צמת לא קטן מהערכים בבניו. זמן הריצה של האלגוריתם O(n) (n הוא מספר הצמתים בעץ).

תרגיל 5

כתבי פונקציה לא רקורסיבית שמקבלת מערך *a* שמייצג ערמה. וסורקת את הערמה בסדר תחילי.

רמז: השתמשי במחסנית

תרגיל 6

ממשי מחלקת ערמה גנרית: תכונות מערך(מסוג T - שחייב להיות IComparable) , heap_size. בנאים: 1. מקבל גודל ומאתחל את המערך להיות בגודל הנדרש 2. מקבל מערך ובונה ממנו ערמה. פונקציות: הוספת ערך, מחיקת המקסימום, החזרת ערך המקסימום, שינוי ערך איבר i (כמובן שיש למשש גם את כל פונקציות העזר הנדרשות, שימי לב: השוואת הערכים תהיה על ידי CompareTo)

כותרת המחלקה תהיה: class Heap<T> where T:IComparable

הגדירי ב main ערמה של מחרוזות, הוסיפי לערמה 10 מחרוזות על ידי קליטה מהמשתמש והדפיסי אותן בצורה ממוינת על ידי קריאה ל extract_max כל עוד גודל הערמה גדול מ 0

תרגיל 7

הגדירי מחלקת ממתינים לרופא: לכל ממתין ישמר שם, שעת הגעה למרפאה, תיאור הבעיה, רמת הדחיפות של הטיפול.

הגדירי את המחלקה כמממשת את IComparable וממשי את פונקצית ההשוואה על פי רמת הדחיפות של הטיפול.

הגדירי מחלקת מרפאה: תכונות ערמת מקסימום המכילה את הממתינים לרופא. מפתח הערמה יהיה רמת הדחיפות

הגדירי את הפונקציות: הוספת מטופל להמתנה, כניסת מטופל, שינוי רמת הדחיפות של מטופל i.

חשבי: כאשר ישנם שני מטופלים ברמת דחיפות זהה, האם ניתן יהיה לתת עדיפות למטופל שהגיע קודם?