Q1. (20%)

A *d*-ary heap is like a binary heap, but (with one possible exception) non-leaf nodes have *d* children instead of 2 children.

How would you represent a *d*-ary heap in an array? Where will the parent and the leftmost son of the node in index *i* be?

ערמה d-ארית דומה לערמה בינרית. ההבדל הוא שבערמה d-ארית לכל צמת יכולים להיות d בנים, ולא רק שני בנים כמו בערמה בינרית.

(הסברי במילים) איך תיוצג ערמה d-ארית במערך? היכן ימצאו הבן השמאלי ביותר והאב של הצמת בעל אינדקס i?

תשובה:

כמו בערמה בינרית, הצמתים יאוחסנו כרצף של ערכים במערך.

The parent of *i* is $\left\lfloor \frac{i+d-2}{d} \right\rfloor$

The leftmost son of *i* is $(i-1)d+2$

Q2. (20%)

Write a C function that gets an array of integers *a* of size *n*, which represents a complete binary tree and prints the pre-order-traversal of the tree.

The header of the function is:

```
void preorder (int a[], int n)
```

The running time is $O(n)$.

כתבי פונקציה ב-C שמקבלת מערך של שלמים a בגודל n, שמייצג עץ בינרי כמעט שלם ומדפיסה סריקה בסדר תחילי של העץ.

כותרת הפונקציה היא:

```
void preorder (int a[], int n)
```

וזמן הריצה שלה $O(n)$.

void preorder (int a[], int n)

תשובה:
הפונקציה קוראת לפונקציה רקורסיבית המבצעת את הסריקה.

```
void preorder (int a[], int n)
{
  recorder(a, n, 0);
}

void recorder (int a[], int n, int i)
{
  if (i<n)
  {
    printf("%i ", a[i]);
    recorder(a,n,2*i+1);
    recorder(a,n,2*i+2);
  }
}
```

Q3. (20%)

A "super AVL tree" is an AVL tree with a pointer to the maximum key and a pointer to the minimum key in the tree.

a. Explain how these pointers can be updated after insertion and deletion of nodes. The running times of these operations should remain $O(\lg n)$.

b. It seems that a "super AVL tree" can do much more than a heap. Give two operations (that were discussed in the MOOC) that can be implemented on a "super AVL tree" more efficiently than on a heap.

c. Give one operation (that was discussed in the MOOC) that can be implemented on a heap more efficiently than on a "super AVL tree".

"עץ סופר AVL" הוא עץ AVL עם מצביע למפתח המקסימלי בעץ ומצביע למפתח המינימלי בעץ.

א. הסבירי איך מצביעים אלה מעודכנים בפעולות הכנסה ומחיקה של צמתים. זמן הריצה של הפעולות צריך להישאר $O(\lg n)$.

ב. נראה ש"עץ סופר AVL" יכול לבצע פעולות מסוימות ביעילות גדולה יותר מאשר ערמה. הביאי דוגמה לשתי פעולות כאלה הנדונות ב-MOOC.

ג. תארי פעולה אחת, שנדונה ב-MOOC, שערמה מבצעת ביעילות גדולה יותר מ"עץ סופר AVL".

תשובה:

א. כדי למצוא מינימום ב AVL יש לרדת מהשורש שמאלה עד שמגיעים לצמת שאין לו בן שמאלי – זה המינימום. כדי למצוא מקסימום ב AVL יש לרדת משורש שמאלה עד שמגיעים לצמת שאין לו בן שמאלי – זה המקסימום. זמן הריצה $O(\lg n)$. ניתן לבצע זאת אחרי כל הכנסה ומחיקה.

ב. חיפוש – ניתן לחפש בסופרAVL בזמן לוגריתמי, בעוד שבערמה זמן החיפוש לינארי. בסופר AVL ניתן למצוא מינימום בזמן קבוע, בעוד שבערמת מקסימום זמן מציאת המינימום הוא לינארי.

ג. בניית ערמה מתבצעת בזמן לינארי בעוד שסיבוכיות זמן הריצה של בניית AVL היא $O(n\lg n)$.

Q4. (20%)

The operations on a "minimum stack" are:

- push(x) – insert x.
- pop( ) – delete the last item that was pushed.
- min( ) – return (but do not delete) the item with the minimal key.

Implement a "minimum stack" using two stacks ("ordinary" stacks):
write an algorithm for each operation (pseudo code).
Explain your implementation.
The running time of every operation is $O(1)$.

הפעולות על "מחסנית מינימום "הן:

push(x) – (הכנס x).

pop( ) – מחק את האיבר האחרון שהוכנס.

min( ) – החזר)אך אל תמחק (את האיבר בעל המפתח המינימלי.

ממשי "מחסנית מינימום "בעזרת שתי מחסניות)מחסניות "רגילות :(")

כתבי אלגוריתם לכל פעולה (פסאודו קוד).

הסבירי את המימוש.

זמן הריצה לכל פעולה הוא $O(1)$.

תשובה:

הרעיון הוא להשתמש במחסנית אחת כמחסנית)להכנסה ומחיקה ,(ובמחסנית השנייה כאחסון למינימום –בראשה תמיד יהיה האיבר המינימלי.
בכל רגע נתון, בשתי המחסניות אותו מספר איברים. ובשתי המחסניות אם נסתכל על איברים באותו המיקום, ב- s1יש את האיבר שהוכנס למחסנית, וב- s2יש את הערך שהיה המינימום כאשר האיבר הוכנס למחסנית s1.

נגדיר פעולת עזר top, שמחזירה)אך לא מוחקת (את ראש המחסנית (וניתנת לשימוש גם על s1 וגם על s2:

```
top()
x←s.pop()
s.push(x)
return x
```

מימוש הפעולות:

```
push(x)
s1.push(x)
if ((empty(s2)) || (s2.top()>x))
    s2.push(x)
else
    then s2.push(s2.top())

pop()
s2.pop()
return s1.pop()

min()
return s2.top()
```

Q5. (20%)

Consider the following algorithm on a list of *n* integers:

1. Insert the integers into a hash table
2. Sort the linked list in every slot
3. Concatenate the linked lists in the slots (in ordered fashion, from slot 0 to slot *m*-1) into one long linked list.

a. What is the expected running time of this algorithm? Expalin.

b. What condition should the hash function satisfy, if we want the linked list created, in step 3, to be a sorted list? Prove your answer.

נדון באלגוריתם הבא, הפועל על רשימה של n שלמים.

1. הכניסי את המספרים לטבלת גיבוב
2. מייני את הרשימה המקושרת בכל תא
3. שרשרי את הרשימות המקושרות זו לזו לפי סדר התאים.

א. מהי תוחלת זמן הריצה (זמן הריצה הממוצע) של האלגוריתם. הסבירי את תשובתך.

ב. ממש

תשובה:

א. צעד 1 מתבצע בזמן קבוע לכל איבר. מכיוון שבממוצע יש $O(1)$ ערכים בכל תא, אז זמן מיון כל רשימה הוא קבוע (לא משנה באיזה אלגוריתם נשתמש). סה"כ זמן הריצה $O(m+n)$ כש-m הוא מספר התאים בטבלה ו-n הוא מספר האיברים.

ב. אם $k_1<k_2$ אז $h(k_1)\le h(k_2)$