

סעיף א':

A=3,2,4,9,11,6

B=5,8,7,1,2

C=12 ידפיס 4 ו3

c=100 ידפיס לא קיים

סעיף א':

func(a,b,m,n,c)

{

sort a with heap sort // mlogm

sort b with heap sort // nlogn

for (i=0;i<m;i++)

{

j=binarySearch(b,n,c-a[i]);

if(j!=-1) {

print(i,j);

return;

}

j=binarySearch(b,n,a[i]-c);

if(j!=-1)

{

print (i,j);

return;

}

}

//mlogn

print("not exist i and j "):

=mlogm+nlogn+mlogn=**O((m+n)log(m+n))**

סעיף ב':

func(a,b,m,n,c)

{

T= new hash table();

//insert all b values into T //O(n) amortized

for (i=0;i<m;i++)

{

j=search(T,c-a[i]);//O(1) amortized

if(j!=null) {

print(i,j);

return;

}

j=search(T,a[i]-c); //O(1) amortized

if(j!=null)

{

print (i,j);

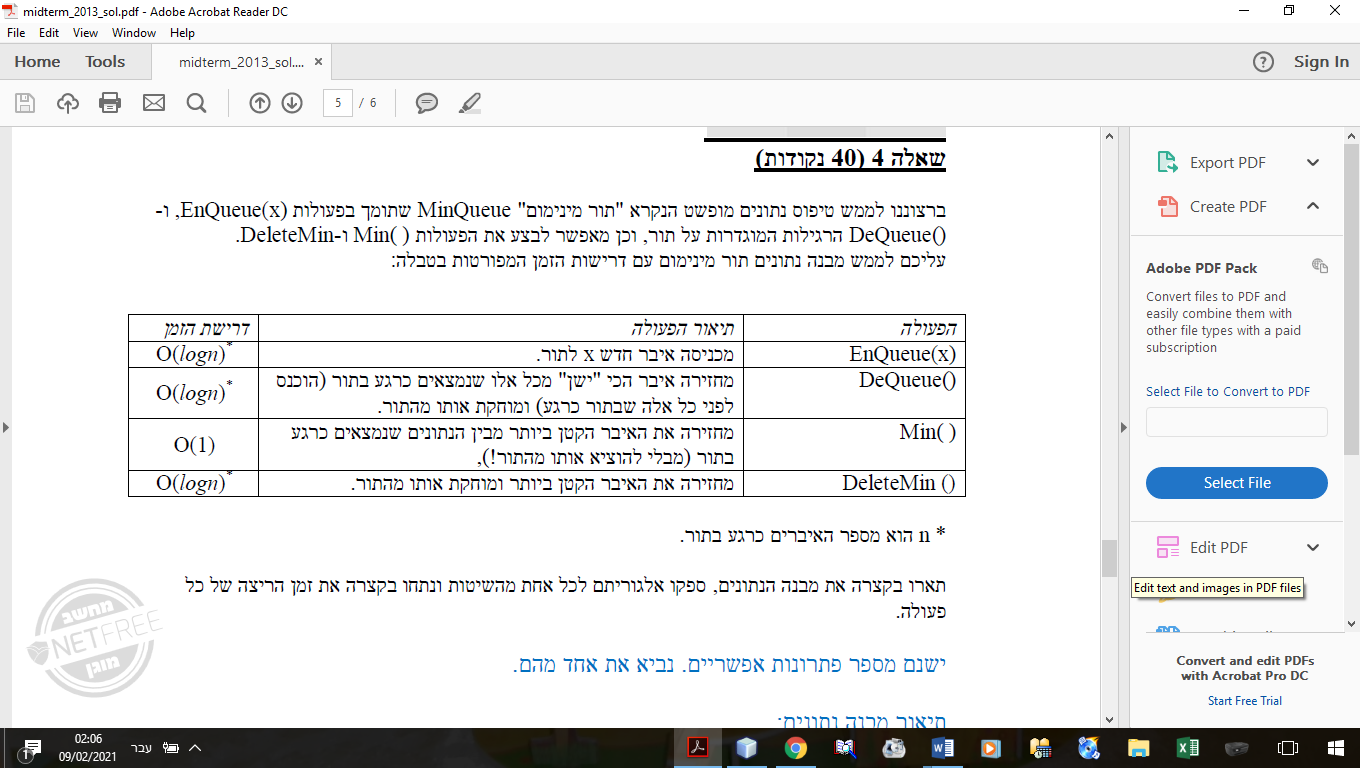
return;

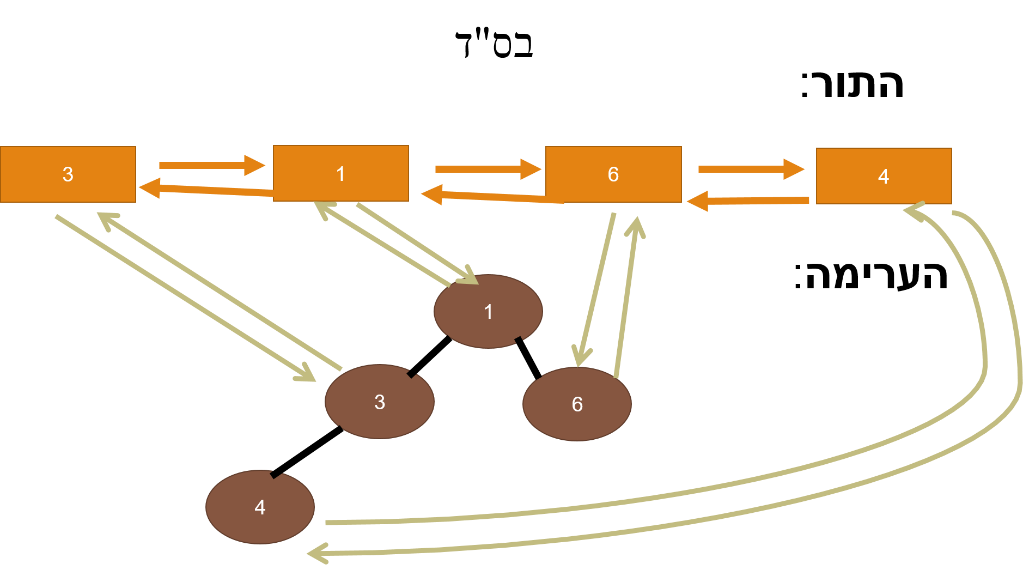
}

}//O(m) amortized

print("not exist i and j "):

=O(m+n)





כדי לממש את המבנה הנתונים נשתמש ברשימה דו כיוונית שתכיל את איברי התור מהישן לחדש כאשר הוספה של איבר תהיה לסוף הרשימה והוצאת איבר ישן מתחילת הרשימה.

בנוסף, נחזיק ערימת מינימום שתכיל את כל איברי המבנה, כאשר כל איבר בערימה מכיל גם מצביע לאיבר הזהה ברשימה ולהיפך לצורך שליפת ומחיקת המינימום – נכנה את המצביע הזה בשם pointer

Enqueue(x) - נכניס איבר חדש לסוף הרשימה הדו כיוונית ונשמור מצביע לאיבר זה בשם L1 -O(1)

נכניס איבר חדש לערימה ונשמור מצביע לאיבר החדש בשם H1 - O(log n)

נעדכן את המצביע pointer של האיבר ברשימה לH1

ונעדכן את המצביע pointer של האיבר בערימה לL1

סה"כ O(logn)

Dequeue – ניגש למצביע pointer של האיבר הראשון ברשימה [שבעצם מצביע לאיבר המתאים בערימה] ונמחוק את האיבר

המוצבע מהערימה –O(logn)

נמחק את ראש הרשימה הדו כיוונית –O(1)

סה"כ:O(logn)

Min() – נחזיר את הערך בראש הערימה

DeleteMin() – ניגש למצביע pointer של ראש הערימה [שמצביע לאיבר הזהה ברשימה] ונמחק את האיבר מהרשימה הדו

כיוונית.

נבצע extarctMin על הערימה ונשמור את הערך המוחזר . O(logn)

נחזיר את הערך ששמרנו.

