Seminar 4 – Priority Queue

1. Sa hapësirë shtesë duhet për të ekzekutuar algoritmin Heapsort?
   a. O(logn)
   b. O(n)
   c. $O(n^2)$
   d. O(1)

2. Cili është kompleksiteti në rastin më të keq në kohë për algoritmin HeapSort nëse inputi është një vektor me n rekorde ku të gjithë vlerat janë unike?
   a. O(logn)
   b. O(nlogn)
   c. $O(n^2)$
   d. O(n)

3. Diskutoni mbi rastet e best, average dhe worst case në HeapSort për një vektor me gjatësi n. Çfarë ndodh nëse kemi si input një vektor i cili i ka të gjitha vlerat e barabarta?

4. Pse nuk perdoret a[0] në ruajtjen e elementëve të një Heap?

5. Cili është numri minimal i elementëve që duhet të shkëmbehen gjatë thirrjes së funksionit *remove_maximum* në një Heap me madhësi *n* në të cilën nuk ka vlera duplikate? Jepni një rast ku arrihet ky numër minimal.

6. **Diskutoni rreth çështjes në vijim:** Gjatë implementimit të një funksioni për gjetjen e maksimumit, për të ruajtur një kohë konstante ekzekutimi, një mënyrë e vlefshme do të ishte mbajtja e vlerës maksimale gjatë shtimeve, dhe më pas kjo vlerë të kthehet si maksimum. A do të sillte vështirësi në implementim?

7. Në klasën për Max Priority Queue shtoni një metodë për resize ose ndryshimin e madhësisë së vektorit.

8. Ndërtoni një algoritëm me kohë lineare që kontrollon nëse një vektor i dhënë është një max heap.

9. Implementoni një metodë ***min()*** e cila duhet të shtohet në implementimin e Max Priority Queue. Implementimi i kësaj metode duhet të ketë një kohë ekzekutimi konstante dhe hapësirë shtesë konstante.

10. Index priority queue. In many applications, it makes sense to allow clients to refer to items that are already on the priority queue. One easy way to do so is to associate a unique integer index with each item. Moreover, it is often the case that clients have a universe of items of a known size N and perhaps are using (parallel) arrays to store information about the items, so other unrelated client code might already be using an integer index to refer to items. These considerations lead us to the following methods:

```
public class IndexMaxPQ<Key extends Comparable<Key>> implements
Iterable<Integer> {
      public IndexMaxPQ(int maxN)
      public void insert(int i, Key key)
      private boolean less(int i, int j)
      private void exch(int i, int j)

      private void validateIndex(int i)
      public boolean contains(int i)
      public int maxIndex()
      public int delMax()
      public void changeKey(int i, Key key)
}
```

Implement the basic operations in the index priority-queue API by modifying the MaxPQ algorithm. The necessary changes are:

a. Change pq[] to hold indices
b. Add an array keys[] to hold the key values,
c. Add an array qp[] that is the inverse of pq[] — qp[i] gives the position of i in pq[] (the index j such that pq[j] is i).
d. Use the convention that qp[i] = -1 if i is not on the queue
e. Include a method contains() that tests the above condition. You need to modify the helper methods exch() and less() but not sink() or swim().
f. Add maxIndex(), changeKey(), and delete() to your implementation.

11. Supozoni se keni sekuencën e mëposhtme:

    P R I O * R * * I * T * Y * * * Q U E * * * U * E

    (një shkronjë nënkupton veprimin e shtimit, ndërkohë shenja e * thirrjen e funksionit remove_max())

    Shkruani sekuencën e vlerave të cilat do të kthehen nga funksionet remove_max()

12. **Dynamic-median finding.** Design a data type that supports *insert* in logarithmic time, *find the median* in constant time, and *remove the median* in logarithmic time.

13. Na jepet një matricë me **n** elementë. Duam të heqim **k** elementët më të vegjël nga kjo matricë. Supozoni se **k, n** janë numra të plotë pozitivë edhe dimë që **n** është shumë më e madhe se **k**.

a. Cilën strukturë të dhënash që kemi mësuar mund të përdorim në këtë rast për të patur një zgjidhje sa më efiçente?
b. Jepni algoritmin (me pseudocode ose ne Java) që zgjidh problemin duke përdorur strukturën që keni dhënë si përgjigje në pikën (a).
c. Analizoni kompleksitetin në kohë për zgjidhjen tuaj.

14. Supozim: Gjithë veprimet në priority queue kërkojnë një kohë konstante. Sa është kompleksiteti në kohë (Big O) për algoritmin Heapsort në rastin më të keq? Shpjegoni përgjigjen.
$O(N)$