KODLAR

MAİN

```
public class Main {


    public static void main(String[] args) {
        int insanSayısı = 10;
        soru1(insanSayısı);
        soru3(insanSayısı);
        soru2(insanSayısı);
    }

    public static void soru2(int insanSayısı) {

        System.out.println("ÖDEV B");
        Random random = new Random();

        MinHeap minHeap = new MinHeap(insanSayısı);

        for (int i = 0; i < insanSayısı; i++) {
            int kuyrukSüre = 0;
            int işlemSüre = (int) (30 + random.nextDouble() * 270);

            int beklenenSüre = işlemSüre + kuyrukSüre;
            minHeap.insert(new Person(beklenenSüre, kuyrukSüre, işlemSüre));
        }
        minHeap.minHeap();

        Person öncekiKişi = minHeap.remove();
        int oncekiBeklemeSuresi = öncekiKişi.beklenenSüre;
        System.out.println("Toplam süre: " + öncekiKişi.beklenenSüre + " İşlem
süresi: " + öncekiKişi.beklenenSüre);

        while (minHeap.Heap.length != 0) {
            Person person = minHeap.remove();
            if (person != null) {
                person.beklenenSüre = person.işlemSüre + oncekiBeklemeSuresi;
                oncekiBeklemeSuresi = person.beklenenSüre;
                System.out.println("Toplam süre: " + person.beklenenSüre + " İşlem
süresi: " + person.işlemSüre);
            }
        }
    }

    public static void soru3(int insanSayısı) {

        System.out.println("ÖDEV C");

        Random random = new Random();
        PriorityQueue<Person> heap = new PriorityQueue<Person>(new
Comparator<Person>() {
            @Override
            public int compare(Person birinciKişi, Person ikinciKişi) {
                return Integer.compare(birinciKişi.işlemSüre,
ikinciKişi.beklenenSüre);
            }
```

```
        });

        PriorityQueue<Person> tempHeap = new PriorityQueue<Person>(new
Comparator<Person>() {

            @Override
            public int compare(Person birinciKişi, Person ikinciKişi) {
                return Integer.compare(birinciKişi.işlemSüre,
ikinciKişi.beklenenSüre);
            }
        });

        for (int i = 0; i < insanSayısı; i++) {
            int kuyrukSüre = 0;
            int işlemSüre = (int) (30 + random.nextDouble() * 270);

            int beklenenSüre = işlemSüre + kuyrukSüre;

            heap.offer(new Person(beklenenSüre, kuyrukSüre, işlemSüre));
            tempHeap.offer(new Person(beklenenSüre, kuyrukSüre, işlemSüre));
        }

        Iterator<Person> iterator = heap.iterator();
        Person öncekiKişi = heap.poll();
        System.out.println("Toplam süre: " + öncekiKişi.beklenenSüre + " İşlem
süresi: " + öncekiKişi.işlemSüre);
        int oncekiBeklemeSuresi = öncekiKişi.beklenenSüre;
        while (iterator.hasNext()) {
            Person person = heap.poll();
            if (person != null) {
                person.beklenenSüre = person.işlemSüre + oncekiBeklemeSuresi;
                oncekiBeklemeSuresi = person.beklenenSüre;
                System.out.println("Toplam süre: " + person.beklenenSüre + " İşlem
süresi: " + person.işlemSüre);
            }
        }
    }

    public static void soru1(int insanSayısı) {

        System.out.println("ÖDEV A");

        Random random = new Random();
        Queue<Person> queue = new LinkedList();
        Queue<Person> tempQueue = new LinkedList();

        for (int i = 0; i < insanSayısı; i++) {
            int kuyrukSüre = 0;
            int işlemSüre;

            if (!queue.isEmpty())
                kuyrukSüre = tempQueue.poll().beklenenSüre;

            int rastgeleSayı = (int) (random.nextDouble() * 270);
            işlemSüre = 30 + rastgeleSayı;
            int beklenenSüre = işlemSüre + kuyrukSüre;
            queue.offer(new Person(beklenenSüre));
            tempQueue.offer(new Person(beklenenSüre));
        }
```

```
        Iterator iterator = queue.iterator();
        while (iterator.hasNext()) {
            Person tempPerson = queue.poll();
            System.out.println("Toplam süre: " + tempPerson.beklenenSüre);
        }
    }
}
```

PERSON

```
public class Person {
    public int beklenenSüre;
    public int kuyrukSüre;
    public int işlemSüre;


    public Person(int beklenenSüre, int kuyrukSüre, int işlemSüre) {
        this.beklenenSüre = beklenenSüre;
        this.kuyrukSüre = kuyrukSüre;
        this.işlemSüre = işlemSüre;
    }

    public Person(int beklenenSüre) {
        this.beklenenSüre = beklenenSüre;
    }
}
```

MİNHEAP

```
class MinHeap {
    public Person[] Heap;
    private int size;
    private int maxsize;

    private static final int FRONT = 1;

    public MinHeap(int maxsize) {
        this.maxsize = maxsize;
        this.size = 0;
        Heap = new Person[this.maxsize + 1];
        Heap[0] = new Person(0, 0, 0);
    }

    private int parent(int pos) {
        return pos / 2;
    }

    private int leftChild(int pos) {
        return (2 * pos);
    }

    private int rightChild(int pos) {
        return (2 * pos) + 1;
```

```
    }

    private boolean isLeaf(int pos) {
        if (pos >= (size / 2) && pos <= size) {
            return true;
        }
        return false;
    }

    private void swap(int fpos, int spos) {
        Person tmp;
        tmp = Heap[fpos];
        Heap[fpos] = Heap[spos];
        Heap[spos] = tmp;
    }

    private void minHeapify(int pos) {

        if (!isLeaf(pos)) {
            if (Heap[pos].işlemSüre > Heap[leftChild(pos)].işlemSüre
                    || Heap[pos].işlemSüre > Heap[rightChild(pos)].işlemSüre) {

                if (Heap[leftChild(pos)].işlemSüre <
Heap[rightChild(pos)].işlemSüre) {
                    swap(pos, leftChild(pos));
                    minHeapify(leftChild(pos));
                } else {
                    swap(pos, rightChild(pos));
                    minHeapify(rightChild(pos));
                }
            }
        }
    }

    public void insert(Person element) {
        if (size >= maxsize) {
            return;
        }
        Heap[++size] = element;
        int current = size;

        while (Heap[current].işlemSüre < Heap[parent(current)].işlemSüre) {
            swap(current, parent(current));
            current = parent(current);
        }
    }

    public void minHeap() {
        for (int pos = (size / 2); pos >= 1; pos--) {
            minHeapify(pos);
        }
    }

    public Person remove() {
        if (Heap[FRONT] == null || Heap[size-1] == null) {
            return null;
        }
        Person popped = Heap[FRONT];
        Heap[FRONT] = Heap[size--];
```

```
        minHeapify(FRONT);
        return popped;
    }
}
```

EKRAN ÇIKTISI:

```
SORU1
Toplam süre: 191
Toplam süre: 285
Toplam süre: 397
Toplam süre: 543
Toplam süre: 575
Toplam süre: 802
Toplam süre: 1062
Toplam süre: 1270
Toplam süre: 1438
Toplam süre: 1481
```

```
SORU2
Toplam süre: 36 İşlem süresi: 36
Toplam süre: 121 İşlem süresi: 85
Toplam süre: 218 İşlem süresi: 97
Toplam süre: 340 İşlem süresi: 122
Toplam süre: 472 İşlem süresi: 132
Toplam süre: 638 İşlem süresi: 166
Toplam süre: 845 İşlem süresi: 207
Toplam süre: 1125 İşlem süresi: 280
Toplam süre: 1409 İşlem süresi: 284
Toplam süre: 1671 İşlem süresi: 262
```

```
SORU3
Toplam süre: 53 İşlem süresi: 53
Toplam süre: 116 İşlem süresi: 63
Toplam süre: 211 İşlem süresi: 95
Toplam süre: 309 İşlem süresi: 98
Toplam süre: 433 İşlem süresi: 124
Toplam süre: 631 İşlem süresi: 198
Toplam süre: 848 İşlem süresi: 217
Toplam süre: 1079 İşlem süresi: 231
Toplam süre: 1333 İşlem süresi: 254
Toplam süre: 1630 İşlem süresi: 297
```