1.

```
public static int count(Employee [] empArr,int num)
 {
   int count=0;

          for(int i=0;i<empArr.length;i++)
          {
          if(empArr[i].getSalary()>num)
                    count++;
                  }
           return count;
            }
```

2.

أ.

```
public double average()
 {
  int sum=0;
      for(int i=0;i<this.results.length;i++)
        {
                    sum=sum+this.results[i];
                          }
  return (double)sum/this.results.length;
      }
```

ب.

```
public void addJump(double jump)
{
  for(int i=this.results.length-1;i>0;i--)
          this.results[i]=this.results[i-1];

      this.results[0]=jump;
      }
```

ج.

```
public static void print(Jumper jumper)
{
  double j1,j2,j3;
  System.out.print("Enter three numbers:");
      j1=in.nextDouble();
      j2=in.nextDouble();
      j3=in.nextDouble();
      jumper.addJump(j1);
      jumper.addJump(j2);
      jumper.addJump(j3);
      double avg=jumper.average();

      int count=0;
      for(int i=0;i<jumper.getResults.length;i++)
      {
            if(jumper.getResults[i]>avg)
                        count++;
                              }
              System.out.println(count);
}
```

3.

أ.

```
public boolean range(int min,int max)
{
 return this.price>=min&&this.price<=max;
 }
```

ب.

```
 public boolean addPhone(Phone phone)
 {
  if(num>=this.phones.length)
      return false;
  this.phones[this.num]=phone;
  this.num++;
  return true;
  }
```

ج.

```
 public void print(int min,int max)
 {
  for(int i=0;i<this.num;i++)
          {
           if(!this.phones[i].getIsBroken()&&this.phones[i].range(min,max))
                   System.out.println(this.phones[i].getPhoneNum());

          }
  }
```

4.

```
        public static Node<Integer> numLst(Node<Details> lst, int numDig) {
    int len = size(lst);
    Details[] arr = new Details[len];
    Node<Integer> newLst = null;
    Node<Integer> pos2 = null;
    Node<Details> pos = lst;

    int i = 0;

    while (pos != null) {
        arr[i] = pos.getValue();
        i++;
        pos = pos.getNext();
    }
    Details[] details = sortArray(arr);

    i = arr.length - 1;
    while (i >= 0 && numDig > 0) {
        if (details[i].getAppears() > 0) {
            if (newLst == null) {
                newLst = new Node<Integer>(details[i].getDigit());
                pos2 = newLst;
            } else {
                pos2.setNext(new Node<Integer>(details[i].getDigit()));
                pos2 = pos2.getNext();
            }
            details[i].setAppears(details[i].getAppears() - 1);
        }
        if (details[i].getAppears() == 0) {
            i--;
        }
        numDig--;

    }
    return newLst;

}

        //عملية مساعدة
        //تعيد طول القائمة
        public static int size(Node<Details> lst) {
    int count = 0;
    Node<Details> pos = lst;
    while (pos != null) {
        count++;
        pos = pos.getNext();
    }
    return count;
}
```

```
        //عملية مساعدة تقوم بترتيب المصفوفة من نمط
        //Details
public static Details[] sortArray(Details[] arr)
{
    int n = arr.length;
    Details temp = null;
    for (int i = 1; i <= arr.length - 1; i++) {
        for (int j = 0; j < arr.length - i; j++) {
            if (arr[j].getDigit() > arr[j + 1].getDigit()) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    return arr;
}
```

5.

```
//
public static boolean isRangeQueue(Queue<Integer> q)
    {
        int min = q.head();
        int len = size(q);
        int i = 1;
        Queue<Integer> temp = new Queue<Integer>( );
        while (i < len)
        {
            temp.insert(q.remove());
            i++;
        }
        int max = q.head();
        temp.insert(q.remove());

        while (!temp.IsEmpty())
        {
            if (temp.head() < min || temp.head() > max)
                return false;
            q.insert(temp.remove());
        }
        return true;
    }
```

```
//פעולת עזר-عملية مساعدة تعيد حجم الدور
public static int size(Queue<Integer>q)
{
        int count=0;
        while(!q.isEmpty())
        {
                count++;
                q.remove();
        }
        return count;
}
```

6.

```
public static Node<Integer> longestConsecutiveList(Node<Integer> lst) {
    if (head == null) {
        return null;
    }

    Node<Integer> longestStart = null;
    Node<Integer> longestEnd = null;
    Node<Integer> currentStart = lst;
    Node<Integer> currentEnd = lst;
    Node<Integer> current = lst;

    while (current != null) {
        // ابدأ بالبحث عن العقد(חוליות) المتتابعة من العقدة الحالية
        while (current.getNext() != null && current.getNext().getValue() == current.getValue() + 1) {
            current = current.getNext();
        }//نهاية الحلقة الداخلية
        currentEnd = current; // تحديث currentEnd

        // تحقق مما إذا كانت القائمة الحالية أطول
        if (longestStart == null || (currentEnd.getValue() - currentStart.getValue()) > (longestEnd.getValue() - longestStart.getValue())) {
            longestStart = currentStart;
            longestEnd = currentEnd;
        }

        // انتقل إلى العقدة التالية للبحث
        if (current.getNext() != null) {
            current = current.getNext();
            currentStart = current; // تحديث currentStart
        } else {
            break; // إذا كان العقدة الحالية هو الأخير في القائمة، انهي الحلقة
        }
    }// نهاية الحلقة الخارجية while

    // استرجاع العناصر المتتابعة
    if (longestEnd != null) {
        longestEnd.setNext(null);
    }

    return longestStart;
}
```

7.

أ-

(1) بعد استدعاء العملية، العملية تعيد القيمة:3

(2) بعد استدعاء العملية، العملية تعيد القيمة:999

(3) العملية تفحص إذا كل منازل العدد زوجية او فردية إذا كان كذلك تعيد عدد منازل العدد خلاف ذلك تعيد القيمة 999-.

ب-

(1) بعد استدعاء العملية، العملية تعيد القيمة 4026

(2) العملية تقوم بإيجاد أكبر عدد بالقائمة التي منازله كلها زوجية او او فردية وتعيد قيمتها.

8.

$$\Sigma = \{a, b\}$$

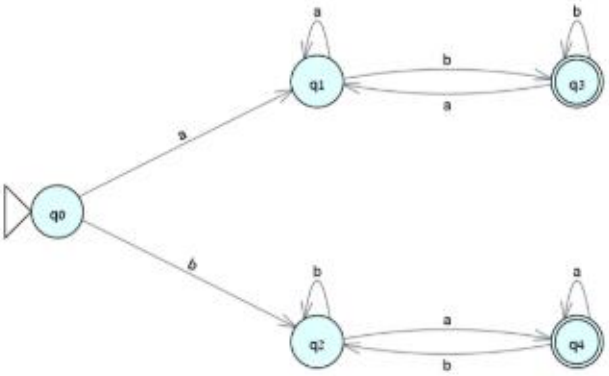
$$L_1 = \{w | w \in \Sigma^*, \#_a(w) = \#_b(w)\}$$

$$L_2 = \{w | w \in \Sigma^*, \#_a(w) > \#_b(w)\}$$

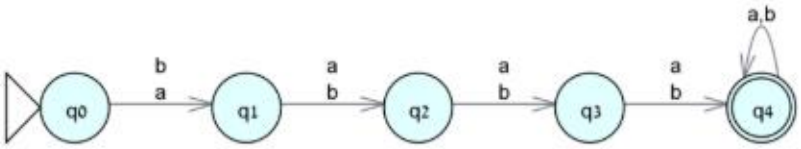
$$L_3 = \{w | w \in \Sigma^*, |w| > 3\}$$

$$L_4 = \{w | w, x \in \Sigma^*, (w = axb \veebar w = bxa)\} = \{a\} \cdot \Sigma^* \cdot \{b\} \cup \{b\} \cdot \Sigma^* \cdot \{a\}$$

א. נבנה אוטומט סופי המקבל את השפה L_4:



ב. נבנה אוטומט <u>סופי</u> המקבל את השפה L_3:



היות וקיימים אוטומטים <u>סופיים</u> המקבלים את השפות L_3 ו-L_4, מכאן שהשפות L_3 ו-L_4 רגולריות.
עפ"י חוקי סגירות, משפחת השפות הרגולריות סגורה תחת <u>חיתוך</u>, מכאן שהשפה $L_3 \cap L_4$ רגולרית גם כן.

ג. נביע את השפה $L_1 \cup L_2$:

$$L_1 \cup L_2 = \{w | w \in \Sigma^*, \#_a(w) \geq \#_b(w)\}$$

השפה $L_1 \cup L_2$ <u>אינה</u> רגולרית, היות וקיימת תלות אינסופית בין כמות ה-a במילה לכמות ה-b במילה.

ד. עפ"י הגדרת משלים ואיחוד, איחוד של שפה עם השפה המשלימה לה הוא השפה האוניברסאלית.
במקרה זה, $L_2 \cup \overline{L_2} = \Sigma^*$.
בנוסף, איחוד כל שפה עם השפה האוניברסאלית הוא השפה האוניברסאלית.
במקרה זה: $L_1 \cup \Sigma^* = \Sigma^*$.

$$L_1 \cup L_2 \cup \overline{L_2}$$
$$L_1 \cup \Sigma^*$$
$$\Sigma^*$$

כלומר: $L_1 \cup L_2 \cup \overline{L_2} = \Sigma^*$.

(אם היינו צריכים להוכיח רגולרית, היינו בונים אסד"מ המקבל את Σ^*: )

ה. בכל המילים בשפה L_1, כמות ה-a במילה **שווה** לכמות ה-b במילה,
אך בכל המילים בשפה L_2, כמות ה-a במילה **גדולה** מכמות ה-b במילה.
מכאן, עפ"י הגדרת החיתוך, בכל המילים בשפת החיתוך $L_1 \cap L_2$, כמות ה-a במילה צריכה להיות גם **שווה** וגם **גדולה** מכמות ה-b במילה. דבר זה אינו אפשרי, כלומר לא קיימות מילים המקיימות את דרישות שתי השפות (L_1 ו-L_2) בו זמנית.
מכאן שהשפות זרות, כלומר $L_1 \cap L_2 = \emptyset$.
השפה הריקה היא שפה סופית (בעלת מספר סופי של מילים, במקרה של השפה הריקה – אפס מילים).
כל שפה סופית היא גם שפה רגולרית, מכאן שהשפה הריקה (ומכאן שגם $L_1 \cap L_2$) רגולרית.

(אם היינו רוצים להוכיח רגולריות באמצעות בניית אסד"מ, היינו בונים אסד"מ המקבל את $\emptyset$: )

ו. תחילה, נביע את השפה $\overline{L_3}$:

$$\overline{L_3} = \overline{\{w | w \in \Sigma^*, |w| > 3\}}$$

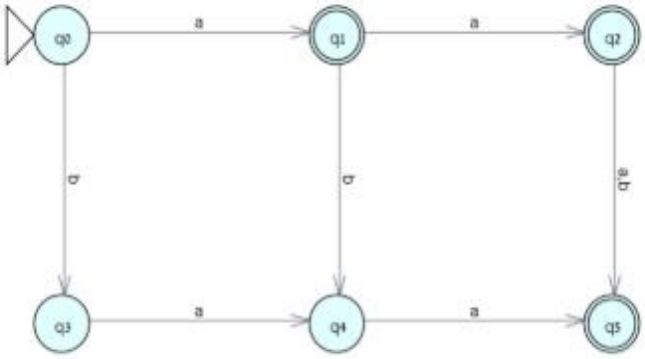
משלים ל-< הוא ≤, נקבל:

$$\overline{L_3} = \{w | w \in \Sigma^*, |w| \leq 3\}$$

כעת נביע את השפה $L_2 \cap \overline{L_3}$:

$$L_2 \cap \overline{L_3} = \{a, a^2, a^3, a^2b, aba, ba^2\}$$

נבנה אוטומט סופי המקבל את השפה $L_2 \cap \overline{L_3}$:



ז. תחילה נביע את השפה $R(L_4)$:

$$R(L_4) = \{w | w, x \in \Sigma^*, (w = bxa \vee w = axb)\}$$

קל לראות כי $R(L_4) = L_4$, מכאן ש-$L_4 \cap R(L_4) = L_4$.