```java
int c=0;                                    //O(1)
for(int i=1; i<n; i++) {                    //O(n) iterations
    for(int j=i; j<=n && j%10 != 3; j++) {  //O(1) iterations
        c++;                                //O(1)
    }
}
```

Hvor mange ganger den innerste for løkken itererer avhenger av i
Her brukes && i sjekken for at for løkken skal stoppe nåe en av de to skjer
j%10 er modulo operator, denne stopper for løkken etter maks 10 iterasjoner

Så totalt blir det O(n)

```java
int s=0;                            //O(1)
for(int i=1; i<n; i++) {            //O(n) iterations
    for(int j=i; j<=n; j++) {       //O(n) iterations
        s = s+1;                    //O(1)
    }
}
```

```java
for(int i=0; i<n; i = i+2) {    //O(n) iterations
    int j = 0;                  //O(1)
    while(j<i)                  //O(n) iterations
        j = j+1;                //O(1)
    int k = 0;                  //O(1)
    while(k<i)                  //O(n) iterations
        k = k+1;                //O(1)
}
```

For løkken øker med 2 hver runde, antall iterasjoner blir fortsatt O(n)
Begge while løkkene går fra 0 til i så det blir O(i) inne i for løkken
Siden det inne i løkken avhenger av i blir det en sum 1+2+3+…+n

Så totalt blir det O(n^2)

```java
for(int i=0; i<n; i++) {    //O(n) iterations
    int j = i;              //O(1)
    while(j>0) {            //O(log n) iterations
        j = j/2;            //O(1)
    }
}
```

For løkken gjør n iterasjoner
while løkken avhenger av i, men gjør maks O(log(n)) iterasjoner
Selv om vi kunne prøvd å regne ut en mer nøyaktig sum
log(1) + log(2) + log(3) + …. Blir det fortsatt samme svar

Så totalt blir det O(n log(n))

```java
int s = 0;
for(int i=1; i<=n; i++) {           //O(n) iterations
    for(int j=2; j<=n; j++) {       //O(n) iterations
        for(int k=3; k<=n; k++) {   //O(n) iterations
            s = i+j+k;              //O(1)
        }
    }
}
```

3 for løkker, selv om de starter på forskjellig tall er alle O(n)

Så totalt blir det O(n^3)

1 Oppgave 1 (20%)

I denne oppgaven skal du gi lavest mulig asymptotiske kjøretid (dvs i O-notasjon) i *verste tilfelle* for ulike operasjoner. Gi alle svar som O(f(N)) uten bruk av mellomrom eller andre tegn, feks $O(N^2 \log N)$.

A er en tabell med N objekt som tilfredsstiller comparable-interfacet.

a) Finn et største element i A.	$O(N)$
b) Sorter verdiene i A med en stabil sorteringsalgoritme.	$O(N \log N)$
c) Omordn verdiene i A slik at de utgjør en binær min-heap.	$O(N)$
d) Sett inn verdiene i A i et binært søketre.	$O(N^2)$
e) Sett inn verdiene i A i et 2-3 søketre.	$O(N \log N)$
f) Avgjør om det finnes to like element i A.	$O(N \log N)$
g) Avgjør hvor mange par med like element som finnes i A.	$O(N \log N)$
h) Sett inn verdiene i A i en hash-tabell som bruker kjedete lister (eng *separate-chaining*).	$O(N^2)$

2 Oppgave 2 (20%)

I denne oppgaven skal du gi lavest mulig asymptotiske kjøretid (dvs i O-notasjon) i verste tilfelle for ulike operasjoner. Gi alle svar som O(f(V,E)) uten bruk av mellomrom eller andre tegn, feks $O(V^2 \log E)$.

G er en graf med V noder og E kanter.

I oppgave a) er kantene i G uvektet og urettet.
I oppgave b) og c) er kantene i G vektet og urettet. T er også et kjent minste utspennende tre til G.
I oppgave d), e) og f) er kantene i G uvektet og rettet.
I oppgave g) og h) er kantene vektet og rettet. Kantvektene kan være negative i g), men ikke i h).

a) Avgjør hvor mange komponenter G består av.	$O(V+E)$
b) Beregn et nytt minste utspennende tre etter at alle kant-vekter har økt med 100.	$O(1)$
c) Beregn et nytt minste utspennende tre etter at vekten av en kant har økt med 100.	$O(V+E)$
d) Avgjør om det for et bestemt par med noder (v,w) finnes en sti fra v til w og tilbake igjen til v.	$O(V+E)$
e) Avgjør om det eksisterer minst et par med noder (v,w) slik at det finnes en sti fra v til w og tilbake igjen til v.	$O(V+E)$
f) Avgjør om det for hvert par av noder (v,w) finnes en sti fra v til w og tilbake igjen til v.	$O(V+E)$
g) Avgjør at det ikke finnes en sykel i G slik at summen av vektene til kantene er negativ.	$O(VE)$
h) For en gitt node v, bestem lengden på en korteste sti fra v til samtlige noder w i G og tilbake igjen til v.	$O(E \log V)$

```java
int findMinGap(ArrayList<Integer> numbers) {
    int n = numbers.size();
    if(n<2)
        throw new IllegalArgumentException("There must be at least 2 numbers in the list

    Collections.sort(numbers);
    int minGap = numbers.get(1)-numbers.get(0);

    for(int i=1; i<n; i++) {
        minGap = Math.min(minGap, numbers.get(i-1)-numbers.get(i));
    }

    return minGap;
}
```

rett svar O(n log n)

- Collections.sort() tar O(n log n)
- For løkken tar O(n)
- Resten tar O(1)
- O(n log n + n + 1) = O(n log n)

```java
static void reverse(ArrayList<String> words) {
    for(int i=words.size()-1; i>=0; i--) {
        words.add(words.remove(i));
    }
}
```

rett svar O(n^2)

Her hadde jeg glemt å skrive at n var words.size()

- For løkken itererer n ganger
- I hver iterasjon brukes både add og remove metoden i ArrayList, add() tar O(1) mens remove tar O(n-i)
- Summen fra n til 1 blir O(n^2)

```java
ArrayList<Double> generateRandom(int n) {
    Random rand = new Random();
    PriorityQueue<Double> stach = new PriorityQueue<>(n);
    ArrayList<Double> output = new ArrayList<>(n);

    while(output.size()<n) {
        stach.add(rand.nextDouble());
        stach.add(rand.nextDouble());
        output.add(stach.poll());
    }
    return output;
}
```

rett svar O(n log n)

- De første linjene oppretter tomme datastrukturer, dette tar O(n) tid
- While løkken kjører n iterasjoner
- Størrelsen på prioritetskøen blir O(n)
- add() og poll() på PriorityQueue tar O(log n) tid
- output.add() tar O(1) tid

```java
static void printFactors(int n) {
    for(int i=1; i<n; i++) {
        System.out.print(i+"=");
        int num=i;
        for(int j=2; j<=Math.sqrt(i); j++) {
            int count=0;
            while(num>1 && num%j==0) {
                num /=j;
                count++;
            }
            if(count>0) {
                if(count>1)
                    System.out.print(j+"^"+count);
                else
                    System.out.print(j);
                if(num>1)
                    System.out.print("*");
            }
        }
        if(num>1)
            System.out.println(num);
        else
            System.out.println();
    }
}
```

rett svar er $O(n\sqrt{n})$

- ytterste for løkke går O(n) ganger
- innerste for løkke går O(sqrt(n)) ganger.
- Det kan se ut som om while løkken inni skal øke kjøretiden men den vil totalt sett skje færre ganger enn den innerste for løkken. Så denne blir O(1) i snitt.
- System.out.println() er O(1)

```java
public static int consecutivePairs(Collection<Integer> numbers) {
    int count = 0;
    for(int num : numbers) {
        if(numbers.contains(num+1))
            count++;
    }
    return count;
}
```

Du skal for hver type Collection finne ut hva kjøretiden til koden er.
Det skal velges 1 svar per rad, hvert rett svar gir 1 poeng, blank eller feil gir 0 poeng.
Finn de som passer sammen:

	O(1)	O(log n)	O(n)	O(n log n)	O(n^2)	O(n^2 log n)	O(n^3)
ArrayList	○	○	○	○	○	○	○
LinkedList	○	○	○	○	○	○	○
HashSet*	○	○	○	○	○	○	○
PriorityQueue	○	○	○	○	○	○	○
TreeSet	○	○	○	○	○	○	○

* Du kan anta at input er laget slik at hash funksjonen for Integer ikke skaper konflikter.

	O(1)	O(log n)	O(n)	O(n log n)	O(n^2)	O(n^2 log n)	O(n^3)
ArrayList	○	○	○	○	✓	○	○
LinkedList	○	○	○	○	✓	○	○
HashSet*	○	○	✓	○	○	○	○
PriorityQueue	○	○	○	○	✓	○	○
TreeSet	○	○	○	✓	○	○	○

```java
public static List<String> matches(char letter, int k){

    List<String> matching = new ArrayList<>();
    for(String word : WORDS) {
        int count = 0;
        for(int index=0; index<word.length(); index++) {
            if(word.charAt(index) == letter) {
                count++;
            }
        }
        if(count>=k) {
            matching.add(word);
        }
    }
    return matching;
}
```

- Her var en kodesnutt som bygget på Wordle koden.
- Koden går igjennom n ord i listen WORDS
- For hvert ord sjekker den hvor mange ganger en bokstav finnes og eventuelt legger denne til i output.
- Siden ordene er 5 bokstaver lange blir kjøretiden O(5 n) = O(n)

```java
public static int longestConsecutive(TreeSet<Integer> numbers) {

    int longestSeq = 0;

    for (int num : numbers) {
        if (num==Integer.MIN_VALUE || !numbers.contains(num - 1)) {
            int currentSeq = 1;

            while (num<Integer.MAX_VALUE && numbers.contains(num + 1)) {
                num++;
                currentSeq++;
            }
            longestSeq = Math.max(longestSeq, currentSeq);
        }
    }
    return longestSeq;
}
```

Oppgave 4

- Vi definerer en gruppe av tall til å være en maximal mengde med tall som kommer etter hverandre uten hull av tall som mangler.
  - If setningen blir true hvis tallet er det første i sin gruppe
  - While løkken går helt til siste element i gruppen
- Det vil si at antall ganger while løkken kaller contains n ganger totalt.
- RETT SVAR: O(n log n)

```java
public double intrestOnLoan(double amount, int n) {

    for(int i=0; i<n; i++) {
        amount = amount * 1.01; //add one percent to amount
    }
    return amount;
}
```

- Her er det en enkel for løkke som itererer n ganger
- Å gange to double tall er O(1)
- Totalt blir det O(n)

```java
public static int countOneBits(int n) {
    int bits = 0;
    while(n>0) {
        if(n%2==1)
            bits++;
        n=n/2;
    }
    return bits;
}
```

- Her er det en while løkke som starter på n
- hver iterasjon deles n på 2 så totalt antal iterasjoner blir O(log(n))
- Operasjonene som gjøres inne i løkken er O(1)
- Totalt blir det O(log(n))

```java
public static int countSteps(int n) {

    int pow = 2;
    int steps = 0;
    for(int i=0; i<n; i++) {
        if(i==pow) {
            pow *= 2;
            for(int j=0; j<n; j++) {
                steps++;
            }
        }
        else {
            steps++;
        }
    }
    return steps;
}
```

- For løkken går O(n) iterasjoner
- If delen skjer hver gang i = 2^k som er O(log(n)) ganger som hver er O(n)
- Else delen skjer O(n-log(n)) ganger som hver er O(1)
- Total kjøretid blir O(n log(n))

```java
public static String makeRandomString(int n) {
    String ans = "";
    for(int i=0; i<n; i++) {
        char c = (char) ('a'+26*Math.random());
        ans += c;
    }
    return ans;
}
```

- Enkel for løkke som bygger en random String av lengde n
- For løkken gjør O(n) iterasjoner
- Å lage en random char er O(1)
- Å legge til en bokstav på slutten av en streng med lengde i tar O(i)
- 1+2+3….+n = O(n^2)

```java
//n = y.size()
public static double computeAreaUnderCurve(LinkedList<Double> y) {
    Double area = 0.0;

    for(int i=1; i<y.size(); i++) {
        area = area + (y.get(i-1)+y.get(i))/2;
    }
    return area;
}
```

- Her er det er for løkke som går igjennom en LinkedList som er O(n) iterasjoner
- Hver iterasjon brukes get() metoden som i LinkedList tar O(min(i,n-1))
- Totalt: 1+2+3…n/2-1+n/2+n/2-1…+3+2+1 = O(n^2)

```java
//n = list.size()
public static Collection<Integer> findLargestK(ArrayList<Integer> list, int k) {
    PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
    for(int num : list) {
        if(pq.size()<k || pq.peek()<num)
            pq.add(num);
        if(pq.size()>k)
            pq.poll();
    }
    return pq;
}
```

- Her er en for løkke som går igjennom alle tallene i en liste, O(n)
- Inne i listen legges det til og/eller trekkes ut av en prioritetskø
- Vi ser at prioritetskøen aldri blir større enn k
- Hver iterasjon av for løkken tar O(log(k)) totalt O(n log(k))