Item 1

I det følgende refererer $\log\ n$ til logaritmen med base 2, dvs. 2-tals-logaritmen. Vær opmærksom på at de sidste tre spørgsmål involverer *Big-Theta* eller *Big-Omega*, og de foregående spørgsmål involverer *Big-O*. Bemærk, i denne opgave skal du have mere end halvdelen rigtige for at opnå point.

In the following, $\log\ n$ refers to log base 2, i.e. the binary logarithm. Be aware that the final three sub-problems involve *Big-Theta* or *Big-Omega*, and the previous sub-problems involve *Big-O*. Note, in this assignment you must have more than half correct in order to obtain points.

	True	False
$3n^{3/2} + \log n = O\left(n^{2/3}\right)$	○	○
$2(\log n)^4 = O\left(n^2\right)$	○	○
$\sqrt{n} \cdot \log n = O(n)$	○	○
$6n^{3/2} = O\left(8^{\log n}\right)$	○	○
$4 \log\left(n^6\right) = O\left((\log n)^2\right)$	○	○
$2^{2 \log n} = O(\log(n!))$	○	○
$n^2 = O\left(n^{3/2}\right)$	○	○
$n^{0.1} = O(n)$	○	○
$n = O\left(n^{1/3}\right)$	○	○
$\sqrt{n} = \Theta(n \cdot \log n)$	○	○
$n^{2/3} = \Omega(n)$	○	○
$n^2 = \Theta\left(n^{0.1}\right)$	○	○

```
Algorithm loop1(n)              Algorithm loop2(n)
  s = 1                           i = 0
  for i = 1 to n                  j = n
    for j = 1 to i                while i < j
      for k = j to i                i = i + 2
        s = s + 1                   j = j + 1
```

```
Algorithm loop3(n)              Algorithm loop4(n)
  i = 1                           i = n
  j = n                           j = 0
  while i < j                     while i > 0
    i = 2 * i                       if j < i
    j = j + n                         j = j + 1
                                    else
                                      j = 0
                                      i = i - 1
```

Angiv udførelses-tiden for hver af ovenstående algoritmer som funktion af n i Θ -notation.

```
Algorithm loop1(n)              Algorithm loop2(n)
  s = 1                           i = 0
  for i = 1 to n                  j = n
    for j = 1 to i                while i < j
      for k = j to i                i = i + 2
        s = s + 1                   j = j + 1
```

```
Algorithm loop3(n)              Algorithm loop4(n)
  i = 1                           i = n
  j = n                           j = 0
  while i < j                     while i > 0
    i = 2 * i                       if j < i
    j = j + n                         j = j + 1
                                    else
                                      j = 0
                                      i = i - 1
```


For each of the above algorithms, state its execution time as a function of n in Θ-notation.

	$\Theta\left(n^3\right)$	$\Theta\left((\log n)^2\right)$	$\Theta(\sqrt{n})$	$\Theta\left(n^2\right)$	$\Theta(n \log n)$	$\Theta(n\sqrt{n})$
loop1	○	○	○	○	○	○
loop2	○	○	○	○	○	○
loop3	○	○	○	○	○	○
loop4	○	○	○	○	○	○

Den følgende kodeudsnit beregner potensen af x. Vi får to heltal, og algoritmen returnerer et heltal. Find den **mindste** Store-O tidskompleksitet for algoritmen.

```
public static long power(int x, int n)
{
    long pow = 1L;

    // loop till n become 0
    while (n > 0)
    {
        // if n is odd, multiply the result by x
        if ((n & 1) == 1) {
            pow *= x;
        }

        // divide n by 2
        n = n / 2;

        // multiply x by itself
        x = x * x;
    }
```

The following code snippet calculates the power of x . We are given two integers, and the algorithm returns an integer. Find the **tightest** Big-O time complexity of the algorithm.

```
public static long power(int x, int n)
{
    long pow = 1L;

    // loop till n become 0
    while (n > 0)
    {
        // if n is odd, multiply the result by x
        if ((n & 1) == 1) {
            pow *= x;
        }

        // divide n by 2
        n = n / 2;

        // multiply x by itself
        x = x * x;
    }
```

A $\mathcal{O}(n)$

B $\mathcal{O}(\log n)$

C $\mathcal{O}(n^2)$

D $\mathcal{O}(n \log n)$

E $\mathcal{O}(n^3)$

F $\mathcal{O}((\log n)^2)$

Item 4

a. Givet et heltalsarray af størrelse N, vil vi kontrollere, om arrayet er sorteret (enten i stigende eller faldende rækkefølge). En algoritme løser dette problem ved at foretage en enkelt gennemløb af arrayet og kun sammenligne hvert element i arrayet med dets naboer. Den værste tidskompleksitet for denne algoritme er:

---

a. Given an integer array of size *N*, we want to check if the array is sorted (in either ascending or descending order). An algorithm solves this problem by making a single pass through the array and comparing each element of the array only with its adjacent elements. The worst-case time complexity of this algorithm is

- [ ] both $\mathrm{O}(N)$ and $\Omega(N)$
- [ ] $\mathrm{O}(N)$ but not $\Omega(N)$
- [ ] $\Omega(N)$ but not $\mathrm{O}(N)$
- [ ] neither $\mathrm{O}(N)$ nor $\Omega(N)$

b. Overvej følgende tre funktioner:

$$f_1 = 10^n, \quad f_2 = n^{\log n}, \quad f_3 = n^{\sqrt{n}}$$

Hvilken af følgende muligheder arrangerer funktionerne i stigende rækkefølge af asymptotisk vækstrate (fx er $\log\ n,\ n,\ n^2$ arrangeret i stigende rækkefølge af asymptotisk vækstrate)?

---

b. Consider the following three functions:

$$f_1 = 10^n, \quad f_2 = n^{\log n}, \quad f_3 = n^{\sqrt{n}}$$

Which one of the following options arranges the functions in the **increasing** order of asymptotic growth rate (e.g. $\log\ n,\ n,\ n^2$ ordered in the increasing order of asymptotic growth rate)?

- [ ] $f_3,\ f_2,\ f_1$
- [ ] $f_2,\ f_1,\ f_3$
- [ ] $f_1,\ f_2,\ f_3$
- [ ] $f_2,\ f_3,\ f_1$