D3 : Tímaflækja og röðun/Time complexity and Sorting

1. (12%) Gefðu einföldustu mögulega tildu-nálgun (~) fyrir eftirfarandi stærðir / Give the simplest possible tilde approximations for the following quantities:

- **The main factor is the largest component in each quantity.**

a. (1 - 1/N)(1 - 2/N)

= **~ 2/N**

b. lg N2 / lg N

= lg N / lg N * lg 2 / lg N

= **1 * lg 2 / lg N**

= lg 2 / lg N

= ~ lg 2 / lg N

c. 2lg N

= **2lg N**

= ~ lg N

2. (15%) Hver er tilda tímaflækja eftirfarandi kóðabúta? / Give the tilde time complexity of each of the following code fragments:

a.
```
int sum = 0;
for (i=0; i < n*n; i++)
    for (j=0; j < 2*n; j++)
        sum++;
```

= **n^2 for the outer loop**

2n for the inner loop

= 4n

b.
```
int sum = 0;
for (int t = n; t > 0; t /= 2)
    for (int i = 0; i < t; i++)
        sum++;
```

= **$n / 2^2$ for the outer loop**

n for the inner loop (or t, which is equal to n)

= n + (n/2 + n/4 + n/8….) **inner loop + (outer loop)**

= n + n

= 2n

c.
```
int sum = 0;
for (i=0; i < n; i++)
    for (j=0; j < i; j+=2)
        sum++;
```

= n **for out loop**

n/2 (or log n) **for inner loop**

= n log n

4. (15%) Ákveðið forrit tekur 2.1 s fyrir inntök af stærð 1000 en 16.8 s fyrir inntök af stærð 4000. Leiddu út tímaflækju reikniritsins á forminu $T(n) = a\ n^b$. / A given program runs in 2.1s on inputs of size 1000 but 16.8s on inputs of size 4000. Derive the time complexity of the program of the form $T(n) = a\ n^b$.

$T(N) = aN^b$

$2,1 = a(1000)^b$ **b = lg(16,8 / 2,1) = lg 8 = 0,9031**

$a = 2,1 / (1000)^{0,9031}$

5. (15%) We are given a randomly ordered array where elements have only three values (small, médium, big), equally many of each type. What is the tilde running time of Insertion sort on this array?

- [medium, big, small, small, big, medium]

N = 6

```
for (int i = 0; i < N; i++)                    → N
    for (int j = i; j > 0; j--)                → ½ N²      (inner loop influenced by outer loop)
        if (less(a[j], a[j-1]))
            exchange(a, j, j-1);
        else break;
```

Tilde running time of insertion sort : ½ N^2

6. (13%) A clerk at a shipping company is charged with the task of rearranging a number of large crates in order of the time they are to be shipped out. Thus, the cost of compares is very low (just look at the labels) relative to the cost of exchanges (move the crates). The warehouse is nearly full – there is extra space to hold any one of the crates, but not two. What sorting method should the clerk use and why (briefly)?

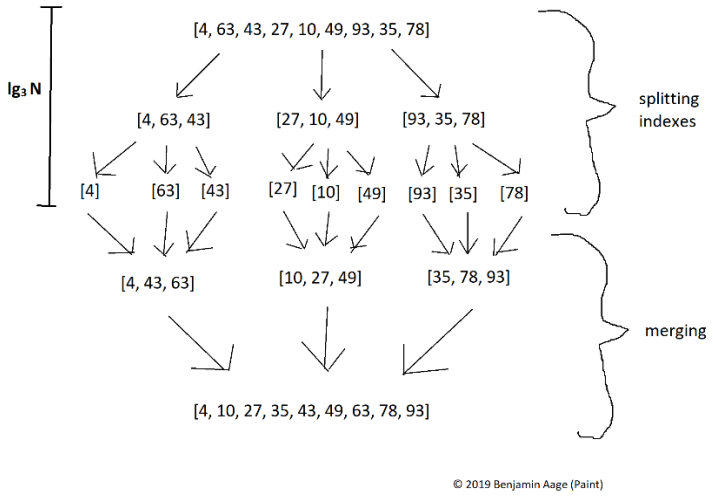
- As the cost of comparison is low compared to the cost of exchange, the sorting method that minimizes the number of swaps would be preferable.

Therefore, it is wiser to use the <u>Selection Sort</u> technique, where in the worst case scenario would be at most n-1 exchanges (n being the number of crates).

Exchanges made in each iteration (at most) would be the exchange between the element (crate) having the smallest value with the current index therefore (and no exchange if the current index has the smallest value already).

7. (15%) Suppose Mergesort is modified to divide the input into three parts, each with one

third of the input. It then sorts each part recursively and combines using three-way merge. Explain why the order of growth of the overall running time is ~(n log n).



v

- By the picture above, by taking the full array (D(N)) at the top and dividing it into three part (D(N/3)) , and thereafter dividing each subpart by three as well (D(N/9)), we get the following:

D(N)	**→**	**N**	**→**	**N**
D(N/3)	**→**	**3(N/3)**	**→**	**N**
D(N/9)	**→**	**9(N/9)**	**→**	**N**

Which leads to the conclusion that:

T(N) = N log N

8. (15%) Explain succinctly how to solve the following problem efficiently: Given two array, determine which elements appear in both arrays.

- The following is an implementation to solve this problem:

Beginning code give:

```
array1 = [F, B, A, D, C, E]

array2 = [Y, C, Z, A, X, W]
```

Implemenatation:

```
pairs = []

for (int i=0; i < (array1.length – 1); i++):

        for (int j = 0; j < (array2.length – 1); j++)

                if array1[i] == array2[j]:

                        pairs.append(array[i])

print(pairs)
```

Output:

```
[A, C]
```