Test 1

Complexity 2011-2012

A. Silva, H. Barendregt, B. Westerbaan & B. Westerbaan

Note: The students who handed in **both** homework assignments can skip Exercise 3 and Exercise 4.

Exercise 1. (2 points) Show that for any real constants a and b, where $a, b > 0$

$$(n + a)^b = \Theta(n^b)$$

Spell out exactly what you are expected to prove (this is worth 1 point) and then prove it (this is worth 1 point).

Answer. We need to prove that there exist $n_0 \in \mathbb{N}$ and $c_1, c_2 \in \mathbb{R}_0$ such that for all $n \geq n_0$

$$c_1 n^b \leq (n + a)^b \leq c_2 n^b \tag{1}$$

First, observe that $n + a > n$, because $a > 0$, and $n + a \leq 2n$ for all $n \geq a$. Moreover, because $b > 0$, these inequalities are preserved by raising both sides to the power b. That is:

$$(n + a) > n^b \quad \text{and} \quad (n + a)^b \leq 2^b n^b \ \ \text{(for all } n \geq a)$$

Hence, take $n_0 = a$, $c_1 = 1$ and $c_2 = 2^b$ and equation (1) holds.

Exercise 2. (4 points) Given an array of unordered numbers, the following function implements an algorithm for finding the k^{th} smallest.

```
int selectKth(int a[], int k, int n){
    int i, j, mini, tmp;
    for (i = 0; i < k; i++){
         mini = i;
         for (j = i+1; j < n; j++)
            if (a[j] < a[mini])
              mini = j;
         tmp = a[i];
         a[i] = a[mini];
         a[mini] = tmp;
   }
      return a[k-1];
}
```

State the complexity of the algorithm in Θ notation in terms of n (and possibly k). Explain your solution. Hint: do a best- and worst-case analysis

Answer. We first label the algorithm with costs and repetitions:

```
int selectKth(int a[], int k, int n){
    int i, j, mini, tmp;                 c1    1
    for (i = 0; i < k; i++){             c2    k+1
         mini = i;                       c3    k
         for (j = i+1; j < n; j++)       c4    Si
            if (a[j] < a[mini])          c5    Sj
              mini = j;                  c6    W
         tmp = a[i];                     c7    k
         a[i] = a[mini];                 c8    k
         a[mini] = tmp;                  c9    k
   }
      return a[k-1];                     c10   1
}
```

The best case occurs when the array is ordered (and $k = 1$, that is we want to find the smallest number). Then, $W = 0$. The worst case occurs when the array is reversely ordered (and $k = n$, that is we want to find the largest number). Then, $W = S_i$.

We first define S_i and S_j:

$$S_i = \sum_{i=0}^{k-1} n - i - 1 = \frac{k}{2}(n - 1 + n - k) = \frac{2kn-k^2-1}{2}$$

$$S_j = \sum_{i=0}^{k-1} n - i = \frac{k}{2}(n + n - k + 1) = \frac{2kn-k^2+1}{2}$$

Then we note that

$$T(n,k) = c_1 + c_{10} + c_2(k+1) + (c_3 + c_7 + c_8 + c_9)k + c_4\frac{2kn - k^2 + 1}{2} + c_5\frac{2kn - k^2 - 1}{2} + c_6 W$$

Hence, in the best case, we replace W by 0 and obtain

$$T(n,k) = c_1 + c_{10} + c_4 + \frac{c_4 - c_5}{2} + c_2(k+1) + (c_3 + c_7 + c_8 + c_9)k + \frac{c_4 + c_5}{2}(2kn - k^2)$$

which is in $\Theta(kn - k^2)$. If we replace k by 1 (see justification above) we obtain

$$T(n,1) = c_1 + c_{10} + c_4 + \frac{c_4 - c_5}{2} + 2c_2 + (c_3 + c_7 + c_8 + c_9) + \frac{c_4 + c_5}{2}(2n - 1)$$

which is in $\Theta(n)$.

In the worst case, not much changes (only c_6):

$$T(n,k) = c_1 + c_{10} + c_4 + \frac{c_4 - c_5 - c_6}{2} + c_2(k+1) + (c_3 + c_7 + c_8 + c_9)k + \frac{c_4 + c_5 + c_6}{2}(2kn - k^2)$$

and hence we still obtain $\Theta(kn - k^2)$. If we replace k by n, we obtain

$$T(n,n) = c_1 + c_{10} + c_4 + \frac{c_4 - c_5 - c_6}{2} + c_2(n+1) + (c_3 + c_7 + c_8 + c_9)n + \frac{c_4 + c_5 + c_6}{2}(2n^2 - n^2)$$

which is in $\Theta(n^2)$.

Exercise 3. (2 points) Consider the following sorting algorithms with an array A as input.

```
void bubble_sort1(int A[], int N) {
  for (i=0 ; i<N ; i++)
    for(j=N ; j>i ; j--)
      if (A[j] < A[j-1])
         swap(A,j,j-1);
}
void swap(int A[], int i, int j) {
  int aux=A[i];
  A[i]=A[j];
  A[j]=aux;
 }


void bubble_sort2(int A[], int N) {
    for (i=0 ; i<N ; i++)
       for(j=N ; j>i ; j--)
          if (A[j] < A[i])
           swap(A,j,i);
}
```

1. Motivate what is the worst case running time of `bubble_sort1`.
2. Idem for `bubble_sort2`.
3. How do these compare to the running time of insertion sort?

Answers.

1. The worst case of `bubble_sort1` occurs when the input is an array sorted in reverse order. In this case the maximum number of swaps will occur. We annotate the function above

```
void bubble_sort1(int A[], int N) {
  for (i=0 ; i<N ; i++)              c1   N+1
    for(j=N ; j>i ; j--)             c2   S1
      if (A[j] < A[j-1])             c3   S2
         swap(A,j,j-1);              c4    W
}
```

   and then write down the function $T(n)$

$$T(n) = c_1(N+1) + c_2S_1 + c_3S_2 + c_4W$$

   The sums S_i and S_j are

$$S_i = \sum_{i=0}^{N-1}(N-i+1) = \frac{N}{2}((N+1)+2) = \frac{N^2+3N}{2}$$
$$S_j = \sum_{i=0}^{N-1}(N-1)-i-1 = \frac{N}{2}((N-2)+(-1)) = \frac{N^2-3N}{2}$$

The worst case occurs when the list is ordered backwards, which has as consequence that the test in the if always succeeds and the swap always occurs ($W = S_j$). Then we have

$$T(n) = Nc_1 + S_i c_2 + S_j(c_3 + c_4).$$

Because S_j and S_i have a N^2 factor, we can conclude that bubble sort executes in the worst case in $\Theta(N^2)$.

The best case is when the vector is sorted and the swap never executes ($W = 0$). We then have

$$T(n) = Nc_1 + S_i c_2 + S_j c_3.$$

but unfortunately the complexity remains quadratic, for the same reason as before. $T(n)$ executes in the best case also in $\Theta(n^2)$.

2. The function `bubble_sort2` performs a similar same amount of operations as `bubble_sort1`, because the two `for` cycles will still be executed no matter what. Hence, it will be in $\Theta(n^2)$.

   Extra: There is however a subtle difference is in the best/worst case analysis. For `bubble_sort1`, the best case occurred when the array was ordered and this will remain the case in `bubble_sort2`. However, whereas in `bubble_sort1` the worst case occurred when the array was sorted in a reverse order, in `bubble_sort2`, the number of swaps in a reverse ordered array is 1 per each inner for (try an example). The worst case for the inner loop, that is when the maximal number of swaps is performed, is if the array is sorted form the second element onwards but the first element is the largest (try, for instance, with `[11,5,6,7,8]`).

3. In the worst case, both insertion sort and bubble sort are in $\Theta(n^2)$. However, in the best case insertion sort behaves linearly, that is, it is in $\Theta(n)$, whereas bubble sort remains quadratic.

Exercise 4. (2 points) Show that the solution of $T(n) = T(n-1) + n$ is in $O(n^2)$.

Answer. We want to show that there exists a constant d such that $T(n) \leq dn^2$ for almost all n. We prove it by induction on n. As mentioned in the slides, we assume $T(n) = O(1)$ for sufficiently small n. And then we reason:

$$\begin{array}{rcll}
T(n) & = & T(n-1) + n & \text{by definition} \\
& \leq & c(n-1)^2 + n & \text{induction hypothesis: } \exists_c\, T(n-1) \leq c(n-1)^2 \\
& = & c(n^2 - 2n + 1) + n & (n-1)^2 = n^2 - 2n + 1 \\
& = & cn^2 - 2cn + n + c & \text{simplification} \\
& \leq & cn^2 - 2cn + 2n & n + c \leq 2n, \text{ if } n \geq c \\
& \leq & cn^2 - 2n^2 + 2n & 2cn \leq 2n^2, \text{ if } n \geq c \\
& \leq & (c-2)n^2 & d = c - 2
\end{array}$$