Exercise Sheet 2
COMS10007 Algorithms 2018/2019

19.02.2019

Reminder: $\log n$ denotes the binary logarithm, i.e., $\log n = \log_2 n$.

1 Proofs by Induction

Prove the following statements by induction:

1. For every integer $n \geq 0$, the following holds:

$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \ .$$

Proof.
Base case: $(n = 0)$
Observe that $\sum_{i=0}^{0} i^2 = 0$ and $\frac{0(0+1)(2\cdot 0+1)}{6} = 0$. The base case thus holds.

Induction Hypothesis: $\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ holds for some value of n.

Induction Step: We need to show that the statement also holds for $n+1$:

$$\begin{aligned}
\sum_{i=0}^{n+1} i^2 &= (n+1)^2 + \sum_{i=0}^{n} i^2 = (n+1)^2 + \frac{n(n+1)(2n+1)}{6} \\
&= \frac{(n+1)\left(6(n+1) + n(2n+1)\right)}{6} = \frac{(n+1)(6n+6+2n^2+n)}{6} \\
&= \frac{(n+1)(n+2)(2n+3)}{6} = \frac{(n+1)(n+2)(2(n+1)+1)}{6} \ .
\end{aligned}$$

□

2. For every $n \geq 1$, the following holds:

$11^n - 6$ is divisible by 5.

Proof.
Base case: $(n = 1)$
Observe that $11^1 - 6 = 5$ which is divisible by 5. The base case thus holds.

Induction Hypothesis: $11^n - 6$ is divisible by 5, for some value of n.

Induction Step: We need to show that the statement also holds for $n + 1$:

$$11^{n+1} - 6 \quad = \quad 11 \cdot 11^n - 6 = 10 \cdot 11^n + 11^n - 6 = 5(2 \cdot 11^n) + 11^n - 6 \ .$$

Observe that the term $5(2 \cdot 11^n)$ is divisible by 5. Furthermore, $11^n - 6$ is divisible by 5 by the induction hypothesis. Since the sum of two numbers that are divisible by 5 is also divisible by 5, the result follows. □

3. Consider the following sequence: $s_1 = 1, s_2 = 2, s_3 = 3$, and $s_n = s_{n-1} + s_{n-2} + s_{n-3}$, for every $n \geq 4$. Prove that the following holds:

$$s_n \leq 2^n \ .$$

Proof.
Base cases: We need to verify that the statement holds for $n \in \{1, 2, 3\}$, since s_n depends on $s_{n-1}, s_{n-2}, s_{n-3}$ (in particular, s_4 depends on s_3, s_2, s_1). This is easy to verify: $s_1 = 1 \leq 2^1, s_2 = 2 \leq 2^2$ and $s_3 = 3 \leq 2^3$.

Induction Hypothesis: We complete the proof using strong induction. The induction hypothesis is therefore as follows: For every $n' \leq n$ the statement $s_{n'} \leq 2^{n'}$ holds.

Induction Step: We need to show that the statement also holds for $n + 1$:

$$s_{n+1} = s_n + s_{n-1} + s_{n-2} \leq 2^n + 2^{n-1} + 2^{n-2} = 2^{n-2}(4 + 2 + 1) \leq 2^{n-2} \cdot 8 = 2^{n+1} \ .$$

□

2 Bubblesort

Bubblesort is a popular, but inefficient, sorting algorithm. It works by repeatedly swapping adjacent elements that are out of order:

Algorithm 1 BUBBLESORT

```
Require: Array $A$ of $n$ integers
1: for $i = 0$ to $n - 2$ do
2:   for $j = n - 1$ downto $i + 1$ do
3:     if $A[j] < A[j - 1]$ then
4:       exchange $A[j]$ with $A[j - 1]$
5:     end if
6:   end for
7: end for
```

1. What is the worst-case runtime of BUBBLESORT?

Proof. Observe that the operation in Line 4, i.e., exchanging two elements in the array, takes time $O(1)$. The runtime is therefore bounded by the number of times Line 4 is executed. The outer loop goes from $i = 0$ to $n - 2$, and the inner loop goes from $j = n - 1$ downto $i + 1$. We therefore compute:

$$\begin{aligned}
\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} O(1) &= O(1) \cdot \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = O(1) \cdot \sum_{i=0}^{n-2} ((n-1) - (i+1) + 1) \\
&= O(1) \cdot \sum_{i=0}^{n-2} (n - i - 1) = O(1) \cdot \left((n-1)^2 - \sum_{i=0}^{n-2} i \right) \\
&= O(1) \left((n-1)^2 - \underbrace{\frac{(n-2)(n-1)}{2}}_{\leq (n-1)^2/2} \right) \leq O(1) \left((n-1)^2/2 \right) \\
&= O(n^2) \ .
\end{aligned}$$

□

2. Consider the loop in lines $2 - 6$. Prove that the following invariant holds at the beginning of the loop:

$$A[j] \leq A[k], \text{ for every } k \geq j \ .$$

Give a suitable termination property of the loop.

Proof.
Initialization: We need to show that the property is true prior to the first iteration of the loop. Let $j = n - 1$. Then the property translates to $A[n-1] \leq A[k]$ for every $k \geq n - 1$. This is trivially true since the only value for k such that $k \geq n - 1$ that also lies within the boundaries of the array is $k = n - 1$. It is of course true that $A[n-1] \leq A[n-1]$. The property thus holds.
Maintenance: Suppose that the property is true before an iteration j of the loop, i.e., $A[j] \leq A[k]$ holds for every $k \geq j$. We will show that the property also holds before the next iteration. Observe that before the next iteration, the value of j is decreased. We thus need to show that after the current iteration, $A[j-1] \leq A[k]$ holds for every $k \geq j - 1$.
Considering the algorithm, there are two cases: Either the if-condition evaluates to true, or it evaluates to false.
Case 1: $A[j] \geq A[j-1]$. (i.e., the if evaluates to false)
In this case nothing happens to the array elements. We need to show that $A[j-1] \leq A[k]$, for every $k \geq j - 1$. We already know that $A[j] \leq A[k]$ for every $k \geq j$. Since $A[j-1] \leq A[j]$, the loop invariant is thus also true.
Case 2: $A[j] < A[j-1]$. (i.e., the if evaluates to true)
In this case, $A[j]$ is exchanged with $A[j-1]$. We need to show that after the exchange $A[j-1] \leq A[k]$ for every $k \geq j - 1$. Consider thus the state of the array after the exchange. Concerning $k = j - 1$, this is trivially true (i.e, $A[j-1] \leq A[j-1]$ clearly holds). Concerning $k = j$, this is also true due to the if-statement evaluating to true and the fact that we exchanged the two elements. Concerning all other values of k, i.e., $k \geq j + 1$, this follows from the loop invariant being true at the beginning of the iteration.
Termination: We are guaranteed that $A[i] \leq A[k]$, for every $k \geq i$. □

3. Consider now the loop in lines $1-7$. Prove that the following invariant holds at the beginning of the loop:

   The subarray $A[0,i]$ is sorted.

   Give a suitable termination property that shows that A is sorted upon termination.

   Proof. We will prove the even stronger statement: "At the beginning of iteration i, the subarray $A[0,i]$ is sorted and $A[0,i-1]$ consists of the $i-1$ smallest elements of A.
   Initialization: We need to show that the property is true prior to the first iteration of the loop. At the beginning of the first iteration we have $i=0$. Then the property translates to "the subarray $A[0\dots 0]$ is sorted and $A[0,-1]$ consists of the $i-1$ smallest elements of A". This is trivially true, since $A[0\dots 0]=A[0]$ consists of a single elements, and $A[0\cdots -1]$ is empty.
   Maintenance: Suppose that the property is true before an iteration i of the loop, i.e., $A[0,\dots,i]$ is sorted and $A[0\dots i-1]$ are the $i-1$ smallest elements of A. We will show that the property also holds before the next iteration. By the termination property stated in the last exercise, we have that $A[i]\leq A[k]$, for every $k\geq i$, or, in other words, $A[i]$ is the smallest element in $A[i,n-1]$. By the loop invariant, $A[0,\dots,i-1]$ are the $i-1$ smallest elements in increasing order. Hence, the subarray $A[0,\dots,i]$ contains the i smallest elements in A in increasing order. This implies further that the subarray $A[0,i+1]$ is sorted (note that no matter which element is at position $i+1$, the array is sorted).
   Termination: We are guaranteed that A is sorted. □

3 More on Big-O

Give formal proofs of the following statements using the definition of Big-O from the lecture.

1. $20n\in O(\frac{1}{4}n^2)$.

   Proof. We have to show that there are constants C,n_0 such that $20n\leq C\cdot\frac{1}{4}n^2$, for every $n\geq n_0$. The previous inequality is equivalent to: $80\leq Cn$. Thus, we can choose $C=1$ and $n_0=80$. □

2. Suppose that $f(n)\in O(g(n))$. Show that: $2^{f(n)}\in O(2^{g(n)})$.

   Proof. This statement is wrong and cannot be proved!
   For example, consider the functions $f(n)=\log n$ and $g(n)=\frac{1}{2}\log n$. Then:

   $$2^{f(n)}=2^{\log n}=n \text{ and } 2^{g(n)}=2^{\frac{1}{2}\log n}=n^{\frac{1}{2}}=\sqrt{n}\ .$$

   Observe that $n\notin O(\sqrt{n})$. The statement is thus false. □

3. Rank the following functions by order of growth: (no proof needed)

   $$(\sqrt{2})^{\log n}, n^2, n!, (\log n)!, (\frac{3}{2})^n, n^3, \log^2 n, \log(n!), 2^{2^n}, n\log n$$

   Hint: Stirling's approximation for the factorial function can be helpful:

   $$e(\frac{n}{e})^n\leq n!\leq en(\frac{n}{e})^n$$

Proof. Following the rough ordering "poly-logarithms" $\leq$ "polynomials" $\leq$ "exponentials", we start with the following ordering (2^{2^n} is even super-exponential!):

$$\log^2 n, n \log n, n^2, n^3, 2^{2^n} \ .$$

The following functions are left:

(a) $(\sqrt{2})^{\log n}$

(b) $n!$

(c) $(\log n)!$

(d) $(\frac{3}{2})^n$

(e) $\log(n!)$

Concerning $(\sqrt{2})^{\log n}$, observe that $(\sqrt{2})^{\log n} = (2^{\frac{1}{2}})^{\log n} = n^{\frac{1}{2}} = \sqrt{n}$. Hence:

$$\log^2 n, (\sqrt{2})^{\log n}, n \log n, n^2, n^3, 2^{2^n} \ .$$

By Stirling's formula, $n!$ is smaller than $n^n = 2^{n \log n}$ and hence $n!$ is in $O(2^{2^n})$:

$$\log^2 n, (\sqrt{2})^{\log n}, n \log n, n^2, n^3, n!, 2^{2^n} \ .$$

Concerning $(\log n)!$, observe that by Stirling's formula, $(\log n)!$ is roughly $(\log n)^{\log n} = 2^{(\log \log n) \log n} = n^{\log \log n}$. Hence, we have

$$\log^2 n, (\sqrt{2})^{\log n}, n \log n, n^2, n^3, (\log n)!, n!, 2^{2^n} \ .$$

Again, by Stirling's formula, we have $(\frac{3}{2})^n \in O(n!)$:

$$\log^2 n, (\sqrt{2})^{\log n}, n \log n, n^2, n^3, (\log n)!, (\frac{3}{2})^n, n!, 2^{2^n} \ .$$

Furthermore, $\log(n!) = \log(n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 2 \cdot 1) = \log(n) + \log(n-1) + \log(n-2) + \dots + \log(2) + \log(1) \leq n \log n$ (this can also be seen by applying Stirling's formula). We can hence insert $\log(n!)$ before $n \log n$:

$$\log^2 n, (\sqrt{2})^{\log n}, \log(n!), n \log n, n^2, n^3, (\log n)!, (\frac{3}{2})^n, n!, 2^{2^n} \ .$$

□

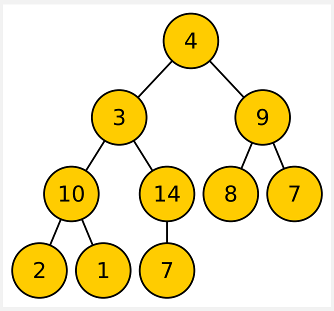
4 Heap Sort

Consider the following array A:

4	3	9	10	14	8	7	2	1	7

1. Interpret A as a binary tree as in the lecture (on heaps).

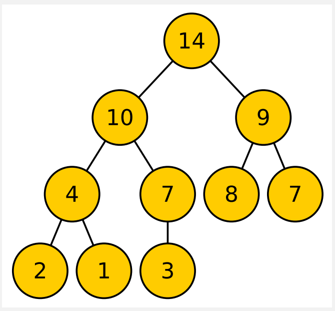
Proof.



□

2. Heapify() the tree. Give the sequence of node exchanges. Draw the resulting heap.

Proof. Apologies, the question should rather be phrased as run Create-Heap() on the initial array. The resulting heap looks as follows:



The sequence of node exchanges are: $14 \leftrightarrow 3$, $3 \leftrightarrow 7$, $4 \leftrightarrow 14$, $4 \leftrightarrow 10$ □

3. What is the worst-case runtime of Heapify() on a binary tree of n elements?

Proof. As discussed in the lecture, Heapify() runs in time $O(\log n)$. This corresponds to the maximum height of a complete binary tree on n elements. □

4. Explain how heap sort uses the heap for sorting. Explain why the algorithm has a worst-case runtime of $O(n \log n)$.

Proof. See lecture. Please let us know (Drop-ins, office hours, etc.) if this is not clear. □

5. Give an array of length n so that heap-sort runs in $O(n)$ time on A.

Proof. For example, an array with $A[i] = 1$, for all $0 \le i \le n-1$. Then Heapify() always runs in time $O(1)$, since no recursive calls are needed. □

5 Merge Sort

This is an open-ended exercise without a single correct solution. This exercise has the potential to lead to further discussions.

Recall the merge operation in Merge Sort: Given is an array A so that the left $\lfloor n/2 \rfloor$ elements and the right $\lceil n/2 \rceil$ elements are sorted in increasing order. The task is to merge the two halves so that the entire array is sorted. The merge operation as discussed in the lecture does not merge in place. Can you think of a merge operation that merges in place? What is the runtime of your suggested merge operation? What is the runtime of merge sort when your merge operation is applied?