Informatics II, Spring 2023, Solution 3

Publication of exercise: March 5, 2023
Publication of solution: March 12, 2023
Exercise classes: March 13 - March 17, 2023

Task 1.

a)

```c
int linear_search(int A[], int n, int t) {
    for(int i = 0; i < n; i++) {
        if (A[i] == t) {
            return 1; // found in the array
        }
    }
    return 0; // not found
}
```

b)

```c
int binary_search(int A[], int n, int t) {
    int l = 0, r = n - 1;
    int mid;
    while (l <= r) {
        mid = (int)((r - l) / 2 + l);
        // printf("%d\n", mid);
        if (A[mid] == t) {
            return 1; // found in the array
        } else if(A[mid] > t) {
            r = mid - 1;
        } else {
            l = mid + 1;
        }
    }
    return 0; // not found
}
```

c) $O(n)$ for `linear_search`. $O(\log_2 n)$ for `binary_search`

d) Run time for linear search grows linearly when n grows.
Run time for binary search grows logarithmically when n grows.

Algorithmic Complexity

Task 2.

a)

Instruction	# of times executed	Cost
result = -1000	1	c_1
for $i := 1$ **to** n **do**	$n+1$	c_2
$current = 0$	n	c_3
for $j := i$ **to** n **by** k **do**	$\frac{n^2-n}{2k} + 2n^{*}$	c_4
$current = current + A[j]$	$\frac{n^2-n}{2k} + n^{**}$	c_5
if $current > result$ **then**	n	c_6
$result = current$	αn^{***}	c_7
return $result$	1	c_8

* $(1+\frac{(n-1)}{k}+1)+(1+\frac{(n-2)}{k}+1)+(1+\frac{(n-3)}{k}+1)+...+(1+\frac{(n-i)}{k}+1)+...+(1+\frac{(n-n)}{k}+1) = \frac{n^2-n}{2k}+2n$

** $(1+\frac{(n-1)}{k}) + (1+\frac{(n-2)}{k}) + (1+\frac{(n-3)}{k}) + ... + (1+\frac{(n-i)}{k}) + ... + (1+\frac{(n-n)}{k}) = \frac{n^2-n}{2k} + n$

*** $\alpha \in [0,1]$

$T(n) = c_1 + c_2(n+1) + c_3 n + c_4(\frac{n(n+1)}{2k}) + c_5(\frac{n(n+1)}{2k} - 1) + c_6(n) + c_7(\alpha n) + c_8$

b) As n gets larger, the leading term in the above formula is n^2. Therefore, the asymptotic complexity of the algorithm is $O(n^2)$.

Asymptotic Complexity

Task 3.

- $f_1(n) = (2n+3)! \in \Theta((2n+3)!)$
- $f_2(n) = 2\log(6^{\log n^2}) + \log(\pi n^2) + n^3 = 2\log n^2 \log 6 + \log \pi + \log n^2 + n^3 = 4 \log 6 \log n + \log \pi + 2\log n + n^3 \in \Theta(n^3)$
- $f_3(n) = 4^{log_2 n} = (2^2)^{log_2 n} = (2^{log_2 n})^2 = n^2 (*) \in \Theta(n^2)$
- $f_4(n) = 12\sqrt{n} + 10^{223} + \log 5^n = 12\sqrt{n} + 10^{223} + n\log 5 \in \Theta(n)$
- $f_5(n) = 10^{\lg 20} n^4 + 8^{229} n^3 + 20^{231} n^2 + 128 n \log n \in \Theta(n^4)$
- $f_6(n) = \log n^{2n+1} = (2n+1)\log n \in \Theta(n \log n)$
- $f_7(n) = \log^2(n) + 50\sqrt{n} + \log(n) \in \Theta(\sqrt{n})$
- $f_8(n) = 14400 \in \Theta(1)$

$$f_8 < f_7 < f_4 < f_6 < f_3 < f_2 < f_5 < f_1$$

(*): hint: $a^{\log_a n} = n$ by the definition.

Special Case and Correctness Analysis

Task 4.

a) The preconditions (inputs) are an array `A[1..n]` and an integer k.

The post conditions(outputs) are the following:

- sum of the biggest k elements of the array `A[1..n]`. Recursively, we can define the output of `algo1` (sum of the biggest k elements of the array `A[1..n]`) in the following way: Let $sum \in \mathbb{N}$ denote the biggest k elements of the array `A[1..n]`, then we have

$$\forall i \in [1..k] : sum = sum{+}A[i], \text{where } A[1..k] \text{ are the biggest } k \text{ integers and sorted in a descending order}$$

- Integers of $A[1..k]$ are the biggest k integers in A and sorted in a descending order.

b) i. The outer loop `for` $i = 1$ **to** k is an `up` loop, as it runs from low (1) to high k.
The inner loop `for` $j = i$ **to** n is an `up` loop as well, as it runs from low (i) to high n.

ii. **Initialization**. $i = 1$ and `A[1..i]` contains only one element.
Maintenance. $i > 1$

$$\forall p \in [i..n], \forall q \in [1..i-1], A[q] \geq A[p]$$

Termination. The loop terminates when $i = k$. `A[1..k]` is sorted in descending order and contains the largest k elements of `A[1..n]`.

c)
- If `A[1..n]` is empty, then the algorithm only initialize `sum` to be zero and returns it.
- If `A[1..n]` only contains one element, the outer loop will be executed only once and guarantees the `A[1..n]` contains the biggest element, which is the only element in the array. The algorithm returns the initialized sum (0) plus the only element in the array ($A[1]$).
- For a general case, the outer loop guarantees that `A[1..n]` contains the biggest k elements. In the body of the outer loop, the algorithm calculates the sum of the first k elements in the array `A[1..n]` and returns it. The returned value is the sum of the biggest k elements.

d)

Instruction	# of times executed	Cost
$sum := 0$	1	c_1
for $i := 1$ **to** k **do**	$k+1$	c_2
$maxi := i$	k	c_3
for $j := i$ **to** n **do**	$\left(kn - \frac{k(k-1)}{2}\right)^{*} + k^{**}$	c_4
if $A[j] > A[maxi]$ **then**	$kn - \frac{k(k-1)}{2}$	c_5
$maxi := j$	$\alpha\left(kn - \frac{k(k-1)}{2}\right)^{***}$	c_6
$sum := sum + A[maxi]$	k	c_7
$swp := A[i]$	k	c_8
$A[i] := A[maxi]$	k	c_9
$A[maxi] := swp$	k	c_{10}
return sum	1	c_{11}

* $(n) + (n-1) + \ldots + (n-(k-1)) = kn - (0+1+...+k-1) = kn - \frac{k(k-1)}{2}$

** k times for terminating loops

*** $0 \leq \alpha \leq 1$

$$T(n) = c_1+c_2(k+1)+c_3k+c_4(kn-\frac{k(k-1)}{2}+k)+c_5(kn-\frac{k(k-1)}{2})+c_6(\alpha(kn-\frac{k(k-1)}{2}))+(c_7+c_8+c_9+c_{10})k+c_{11}$$

In conclusion, $T(n) = k * n$.

e) **Best case**
In the best case, the array has already been sorted in descending order, hence we do not need to run `maxi := j`, i.e., $\alpha = 0$. In this case,

$$T_{\text{best}}(n) = c_1+c_2(k+2)+c_3k+c_4(kn-\frac{k(k+1)}{2}+k)+c_5((k+1)n-\frac{k(k+1)}{2})+0+(c_7+c_8+c_9+c_{10})k+c_{11}$$

$$T_{\text{best}}(n) = O(k * n)$$

Worst case
Similarly, in the worst case, the array is sorted in ascending order and we have to update `maxi` every time, i.e., $\alpha = 1$. In this case, $T_{\text{worst}}(n) = c_1 + c_2(k+2) + c_3k + c_4(kn - \frac{k(k+1)}{2} + k) + c_5((k+1)n - \frac{k(k+1)}{2}) + c_6((k+1)n - \frac{k(k+1)}{2}) + (c_7 + c_8 + c_9 + c_{10})k + c_{11}$

$$T_{\text{worst}}(n) = O(k * n)$$

Asymptotic complexity of best and worst case

$$T_{\text{best}}(n) = O(k * n)$$

$$T_{\text{worst}}(n) = O(k * n)$$

Tasks in past exams

[2021 Final Exam] False.