1. Questions on Big-O Analysis. (25 marks)

1. Consider a polynomial function of order 𝑘 given by 𝑓(𝑛) = 𝑎k𝑛k + 𝑎k-1𝑛k-1 + ∙∙∙ + 𝑎0. Formally demonstrate that 𝑓(𝑛) ∈ Ο(𝑛k). Full marks for using basic definitions and concepts.

[12 marks]

**Big-O Rules: Drop smaller terms**

If f(n) = (1 + h(n)) with h(n) -> 0 as n -> ∞. Then, f(n) is O (1).

**Big-O Multiplication Rules**

Suppose two equations 𝑓1(𝑛) is Ο(𝑔1(𝑛)) and 𝑓2(𝑛) is Ο(𝑔2(𝑛)). From the definition, there exists positive constants c1, c2, n1 and n2 such that 𝑓1(𝑛) ≤ c1𝑔1(𝑛) for all n ≥ n1 and 𝑓2(𝑛) ≤ c2𝑔2(𝑛) for all n ≥ n2. Let n0 = max {n1, n2}. Multiplying 𝑓1(𝑛) and 𝑓2(𝑛) gives 𝑓1(𝑛) 𝑓2(𝑛) ≤ c1c2𝑔1(𝑛)𝑔2(𝑛) for all n ≥ n0. So, 𝑓1(𝑛)𝑓2(𝑛) is O(𝑔1(𝑛)𝑔2(𝑛)).

𝑓(𝑛) = 𝑎k𝑛k + 𝑎k-1𝑛k-1 + ∙∙∙ + 𝑎0

𝑓(𝑛) = nk (𝑎k+ 𝑎k-1/n+ ∙∙∙ + 𝑎0/nk)

As n -> ∞, (𝑎k+ 𝑎k-1/n+ ∙∙∙ + 𝑎0/nk) -> (𝑎k+ 0 + ∙∙∙ + 0) -> ak and 𝑓(𝑛) -> nkak by Drop smaller terms.

O(nk) is trivially O(nk) and O(ak) is O(1).

With Big-O Multiplication Rules, O(𝑓(𝑛)) = O(nkak) = O(nk) O(ak) = O(nk)O(1) = O(nk\*1) = O(nk)

So, 𝑓(𝑛) is O(nk) or 𝑓(𝑛) ∈ Ο(𝑛k).

1. Refer to the definition of Big-O in the lecture materials. In particular, the condition for which one can state that 𝑓(𝑛) is Ο(𝑔(𝑛)) is defined. Briefly explain why the notation 𝑓(𝑛) ∈ Ο(𝑔(𝑛)) is preferred compared to 𝑓(𝑛) = Ο(𝑔(𝑛)). Full marks for using basic definitions and concepts and mathematical formulation.

[13 marks]

𝑓(𝑛) is Ο(𝑔(𝑛)) if and only if there exists positive constants c and n0 such that 𝑓(𝑛) ≤ c𝑔(𝑛) for all n ≥ n0. In mathematical notation, ∃c > 0, n0 such that ∀n ≥ n0, 𝑓(𝑛) ≤ c𝑔(𝑛).

f: N+ -> R+ and g: N+ -> R+ where N+ = {1,2,3, …} and R+ = {x ∈ R| x ≥ 0} assuming that f(n) ≥ 0, ∀n ≥ 1. For convenience, the function is sometimes relaxed to f(n) ≥ 0, ∀n ≥ N for some constant N.

O(n) can be thought of as the set of all functions whose growth is no worse than linear for sufficiently large n. Hence, it can be thought of as the infinite set {1, 2, …, log n, 2 log n, …, n, 2n, 3n, …, n+1, n+2, …}. So, 3n+5 is O(n) is just the statement that 3n+5 is in this set or 3n+5 ∈ O(n).

We should avoid writing big-O notations in the form 𝑓(𝑛) = Ο(𝑔(𝑛)). As an example, if 𝑓(𝑛) is Ο(n), then f(n) is also Ο(n2) because if f(n) grows no worse than linear for sufficiently large n, then f(n) must also grow no worse than quadratic for sufficiently large n. If we write 𝑓(𝑛) = Ο(n) and f(n) = Ο(n2). Then this implies that Ο(n) = Ο(n2) which is incorrect as Ο(n) ⊂ Ο(n2). Instead, we should write 𝑓(𝑛) ∈ Ο(n) and f(n) ∈ Ο(n2).

2. Questions on Binary Search Tree, Heap, Balanced Binary Search Tree, Basic Data Structures (stack/queue)

(a)

(b)

(c)

As the upgrade request and cancellations is in O(log n) time. This will mean that it traverse through the list with a time complexity of O(log n) just to update the customer’s request for upgrade or cancel it’s upgrade (to change its status is O(1). Thus. O(1 + log n) would mean O(log n) (Big-O Rules: Drop smaller terms)). As for the k-highest-priority flyers on the waiting list is O(k log n) time. It would mean that. It undergoes k times as there are k seats available for the upgrade to business class / first class. As the time complexity for traversing the list is O(log n). to under go it k times for upgrading the customers will be O(k \* log n). Thus, O(k log n).

Pseudocode:

Algorithm reqUpgrade(record, customer)

checkRecord = record;

While (checkRecord != null)

If(checkRecord.rank > customer.rank)

checkRecord = checkRecord.right

Else if(checkRecord.rank < customer.rank)

checkRecord = checkRecord.left

Else If(checkRecord.timeReq < customer.timeReq)

checkRecord = checkRecord.right

Else If(checkRecord.timeReq > customer.timeReq

checkRecord = checkRecord.left

Insert(customer)

Balancing(record)

Algorithm removeReq(record, code)

rankCode = code[1]

refNum = code[2] + code[3]

timeReq = code[4] + code[5]

checkRecord = record;

While (checkRecord != Code)

If(checkRecord.rankCode < rankCode)

checkRecord = checkRecord.left

else if(checkRecord. rankCode > rankCode )

checkRecord = checkRecord.right

else if(checkRecord. timeReq > timeReq)

checkRecord = checkRecord.right

else if(checkRecord. timeReq < timeReq)

checkRecord = checkRecord.left

else if(checkRecord.refNum > refNum)

checkRecord = checkRecord.right

else if(checkRecord. refNum < refNum)

checkRecord = checkRecord.left

if(checkRecord == NULL)

return false

delete(checkRecord)

balance(record)

return true