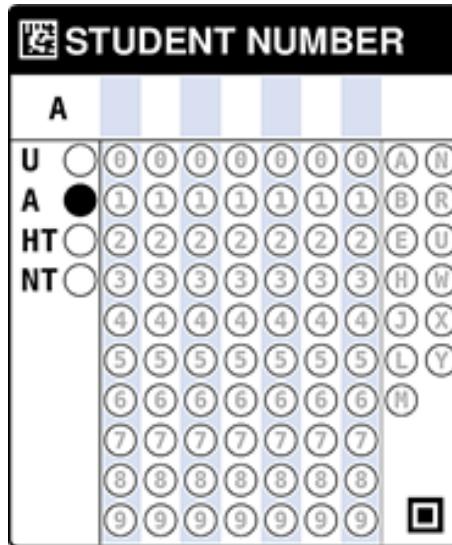


The Answer Sheet for Semester 1 AY2025/26

Write your Student Number in the box below using (2B) pencil. Do NOT write your name.



Section	Maximum Marks	Your Marks	Grading Remarks
Total	100		

Box A.1. Grid Problem Special case 1: return $\text{grid}[0][0] * k$
 Idea:
 Use greedy algorithm. For each row i , we find the $\text{limits}[i]$ largest elements and add them to a candidate list I . Then we take the k largest elements from I and return the sum.

Justification:
 For each row i , we can only take at most $\text{limits}[i]$ elements, and to maximise the sum we should always take the largest elements available. And in the optimal solution, all k selected elements must come from these elements.

Time complexity:
 For convenience, denote $I_i = \text{limits}[i]$ and $s = \text{sum}(\text{limits})$.

(i) Naive approach: We can first sort the m rows each of size n , which is $O(mn \log n)$. Then we sort the candidate list I and take the k largest elements, which is $O(s \log s)$. Thus the overall time complexity is $O(mn \log n + s \log s)$. Use merge sort to get worst case guarantees.

(ii) A slightly better approach: We can maintain min-heaps for selecting the largest elements. For each row i this takes $O(n \log I_i)$. For the candidate list this takes $O(s \log k)$. Thus overall this takes $O(\sum_i n \log I_i + s \log k)$.

(iii) The best approach: Use median of medians (see: https://en.wikipedia.org/wiki/Median_of_medians) when selecting the largest elements. This avoids the logarithmic factor from sorting and guarantees a worst-case time complexity of $O(mn + s)$. Note that QuickSelect does not work here as it is $O(n^2)$ in the worst case. Unfortunately I think this is CS3230 material and beyond IT5003 scope.

Code:
 On the last page.

I claim that my solution runs in $O(\text{mn} + \text{sum}(\text{limits}))$.

Box A.2. A Linked List Task

```

# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next

class Solution:
    # you can create helper function(s)
    def reverse(self, node: Optional[ListNode]) -> Optional[ListNode]:
        pre = None
        cur = node
        while cur:
            nex = cur.next
            cur.next = pre
            pre = cur
            cur = nex
        return pre

    def removeNodes(self, head: Optional[ListNode]) -> Optional[ListNode]:
        # complete this
        if not head or not head.next:
            return head
        head = Solution.reverse(head)
        dum = ListNode(0, head)
        pre = dum
        cur = head
        msf = float('inf')
        while cur:
            if cur.val > msf:
                pre.next = cur.next
            else:
                msf = cur.val
                pre = cur
            cur = cur.next
        return Solution.reverse(dum.next)

```

Testing: see next page

return the updated SLL at the end

I claim that my solution runs in $O(\text{_____} n \text{_____})$.

Box A.3. Three Bags of Candies Special case 1: return $7 * a$
 Special case 2: return $7 * \min(a, b) + 1$

Idea:

Return $7 * \min((a+b+c) // 2, a+b+c - \max(a, b, c))$.

Justification:

Obviously maximising the score is equivalent to maximising the number of moves, M.

Notice that M is bounded by the total number of candies, T, divided by 2, as each move always removes exactly 2 candies.

Also, once all candies outside the largest bag are gone, one cannot continue. Thus M is also bounded by the sum of the number of candies of the two smaller bags, S.

Indeed, if $\max(a, b, c) > S$, then by pairing each of the S candies with the candies from the largest bag, one gets exactly S moves. Otherwise, we are able to use up all candies if T is even, or all but one if T is odd. Thus M is exactly the minimum of S and $T // 2$.

Time complexity:

$O(1)$ because each operation takes constant time, independent of a, b, c.

Code:

```
class Solution:
    from typing import List

    def threeBagsofCandies(self, a: int, b: int, c: int) -> int:
        return 7 * min((a + b + c) // 2, a + b + c - max(a, b, c))

tests = [[1, 3, 4], [1, 2, 4], [7, 7, 1], [7, 5, 1], [2, 3, 5], [2, 2, 2]]
for t in tests:
    print(f"{t} -> {Solution().threeBagsofCandies(*t)}")
```

I claim that my solution runs in $O(\text{_____}^{\textcolor{red}{1}} \text{_____})$.

Box A.4. Feedback about IT5003 S1 AY25/26 changes: Add LeetCode and Midterm

```
# P2 Testing
def build(values):
    head = None
    for v in reversed(values):
        head = ListNode(v, head)
    return head

def to_list(head):
    out = []
    while head:
        out.append(head.val)
        head = head.next
    return out
```

– END OF PAPER; All the Best –

```
tests = [[], [7], [1, 2, 3, 4], [4, 3, 2, 1], [5, 2, 13, 3, 8],
         [2, 2, 2], [10, 5, 10], [1, 1, 1, 2], [9, 1, 2, 3, 0],
         [1, 90, 19, 1, 73, 15, 70]] 7
for t in tests:
    print(f"{t} -> {to_list(Solution().removeNodes(build(t)))}")
```

```

class Solution:
    from typing import List

    def gridProblem(self, grid: List[List[int]], limits: List[int], k: int) -> int:
        def select(a, k): # returns the kth smallest element using median-of-medians, O(n)
            while True:
                n = len(a)
                if n <= 25:
                    return sorted(a)[k]
                m = [sorted(a[i:i + 5])[len(a[i:i + 5]) // 2] for i in range(0, n, 5)]
                p = select(m, len(m) // 2)
                l, h = [_ for _ in a if _ < p], [_ for _ in a if _ > p]
                if k < len(l):
                    a = l
                elif k < n - len(h):
                    return p
                else:
                    k -= n - len(h)
                    a = h
            a = []
            for i, r in enumerate(grid):
                t = limits[i]
                if t == 0:
                    continue
                c = select(r, len(r) - t)
                l = [x for x in r if x > c]
                a += l
                if t - len(l) > 0:
                    a += [c] * (t - len(l))
            if k >= len(a):
                return sum(a)
            c = select(a, len(a) - k)
            l = [x for x in a if x > c]
            return sum(l) + c * (k - len(l))

tests = [[[7, 7, 7], [7, 7, 7]], [1, 3], 2,
         [[[5, 10, 3, 1]], [3], 2],
         [[[1, 2, 3], [4, 5, 6], [7, 8, 9]], [3, 2, 1], 2],
         [[[22, 1, 33], [21, 4, 77]], [2, 3], 3]]
for t in tests:
    print(f"{t} -> {Solution().gridProblem(*t)}")

```