

National University of Singapore  
School of Computing  
**CS4234 - Optimisation Algorithms**  
(Semester 1: AY2021/22)

Date and Time: Thursday, 07 October 2021, 12.02-13.42 (100m)

---

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this midterm test assessment paper until you are told to do so.
2. This assessment paper contains **THREE** (3) sections.  
It comprises **FOURTEEN** (14) printed pages, including this page.  
However, only page 2 to 8 (7 pages) contain questions.  
Page 9-14 (6 pages) are the empty boxes (the answer sheet).
3. This is an **Open Book/Open Laptop Assessment**.  
You are allowed to use your laptop (with Wi-Fi off) as you see fit.  
But you are **NOT** allowed to use the Internet.
4. Answer **ALL** questions within the **boxed space** of the answer sheet.  
**Only if** you need more space, then you can use the empty last page 14.  
You can use either pen or pencil. Just make sure that you write **legibly!**
5. Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.  
Read all the questions first! Some (subtask) questions might be easier than they appear.
6. You can use **pseudo-code** in your answer but beware of penalty marks for **ambiguous answer**.  
You can use **standard, non-modified** classic algorithm in your answer by just mentioning its name, e.g. run Dijkstra's on graph  $G$ , run Kruskal's on graph  $G'$ , etc.
7. All the best :)

## A The Simpler Questions (7x5 = 35 marks)

### A.1 Create Min-Vertex-Cover test case with large $|MVC|$ (5 marks)

In Lecture 01, we learn about the  $O(2^k \times m)$  PARAMETERIZEDVERTEXCOVER algorithm which is still an exponential time algorithm but takes advantage that many VERTEX-COVER instances have small  $|MVC|$ , i.e.,  $k = |MVC|$  is closer to 0 than to  $|V|$ .

But this small  $|MVC|$  is not always the case. So here is the challenge: Draw or describe any simple and small undirected unweighted graph with exactly  $|V| = 7$  vertices so that  $|MVC| > |V|/2$ , i.e., at least  $\lceil 7/2 \rceil = 4$  vertices. Give a short explanation of why your answer satisfies the requirements.

### A.2 Dealing with Min-Vertex-Cover with large $|MVC|$ (5 marks)

If you are presented with MIN-VERTEX-COVER instances with large  $k = |MVC| > |V|/2$  and you are asked to solve all instances optimally and universally (on any general graph), what will you do?

### A.3 Can we do Set-Cover $\leq_p$ Vertex-Cover? (5 marks)

In Lecture 03a, we have seen a simple  $O(V + E)$  polynomial time reduction: Reduce any instance of the proven to be NP-complete problem decision problem of VERTEX-COVER (from Lecture 01) into an instance of decision problem of SET-COVER by setting  $X = E$ ,  $S = V$ , and set edges from vertex  $u \in S$  to all edges  $\in X$  that are covered by  $u$ . The subset(s) taken in the SET-COVER is/are the vert(ices) taken in the VERTEX-COVER. This VERTEX-COVER  $\leq_p$  SET-COVER reduction establishes that SET-COVER is also NP-hard.

Now, can we do the reverse, i.e., reduce any instance of SET-COVER into an instance of VERTEX-COVER (SET-COVER  $\leq_p$  VERTEX-COVER)? If we can, just give a simple argument. If we cannot, also just give a simple argument.

### A.4 Can we do 2-Set-Cover $\leq_p$ Vertex-Cover? (5 marks)

Let 2-SET-COVER be SET-COVER problem where we are restricted to instances where each item appears in exactly two sets, e.g.,  $X = \{1, 2, 3, 4\}$ ,  $S = \{A : \{1, 2\}, B : \{1, 3, 4\}, C : \{3\}, D : \{2, 4\}\}$ .

Now, can we do 2-SET-COVER  $\leq_p$  VERTEX-COVER? If we can, please show the required polynomial time reduction. If we cannot, please show a counter example/explanation.

### A.5 Partition written as Integer-Linear-Program (ILP) (5 marks)

In PS1+PS2+Tutorial 02, we are introduced with the PARTITION problem. The NP-complete decision version is to decide whether a given multiset  $S$  of  $n$  positive integers  $\{s_1, s_2, \dots, s_n\}$  can be partitioned into two subsets  $S_1$  and  $S_2$  such that the sum of the numbers in  $S_1$  equals the sum of the numbers in  $S_2$ . The NP-hard optimization version is to partition the multiset  $S$  into two subsets  $S_1, S_2$  such that the difference between the sum of elements in  $S_1$  and the sum of elements in  $S_2$  is minimized.

Now, express the optimization version of PARTITION as an INTEGER-LINEAR-PROGRAM (ILP)!

### A.6 Partition Puzzle (5 marks)

You are given a multiset  $S = \{167, 247, 323, 387, 331, 132, 222\}$  with  $n = 7$  non-negative integers and you are told that the sum of  $S$ , denoted as  $sum(S) = 1809$ . With this information, answer:

- a. Can we partition  $S$  into two disjoint sets  $S_1$  and  $S_2$  so that their sums are equal? (1 mark)
- b. If your answer is true, then show the content of  $S_1$  and  $S_2$  such that  $sum(S_1) == sum(S_2)$ . If your answer is false, then show the content of  $S_1$  and  $S_2$  below so that the absolute differences of their sums  $|sum(S_1) - sum(S_2)|$  is the minimum possible.

Grading scheme:

3 marks for the  $sum(S_1)$  and  $sum(S_2)$  values  $sum(S_1) \geq sum(S_2)$ , and  
1 mark for the actual content of  $S_1$  and  $S_2$ ...

### A.7 Max-TSP... (5 marks)

In Lecture 04, we learn about the famous TSP ~~problem~~: Find the **mINImum** cost cycle that visits all the vertices of a given complete (**positive**) weighted graph **exactly once**. We then exposed to a few known variants of this famous TSP ~~problem~~.

Now let's explore yet another variant, called the MAX-TSP: Find the **mAXImum** cost cycle that visits all the vertices of a given complete (**positive**) weighted graph **exactly once**. Give a quick thought and write a short argument on whether MAX-TSP is still an NP-hard optimization problem or whether it actually turns into a special case with a polynomial solution.

Note that due to the small marking scheme, there is no need to write a full proof/full polynomial solution (either route) for this question.

## B Multiprocessor-Scheduling (30 marks)

Consider the following MULTIPROCESSOR-SCHEDULING problem: We have  $m$  (identical) processors  $p_1, p_2, \dots, p_m$  and  $n$  tasks  $t_1, t_2, \dots, t_n$ . Each task  $t_i$  has a positive length  $l(t_i)$ . The goal is to assign tasks to processors so as to minimize the maximum load on any processor.

More formally, an assignment is a function  $f : \{1, 2, \dots, n\} \rightarrow \{1, 2, \dots, m\}$  which specifies which task is assigned to which processor. The load on a processor is defined to be:

$$\text{load}(i) = \sum_{j:f(j)=i} l(t_j) .$$

The maximum load is defined as:

$$\text{max-load} = \max_{i \in \{1, 2, \dots, m\}} \text{load}(i)$$

The goal is to find an assignment that minimizes the **max-load**.

### B.1 Sample Test Cases (1+2+2 = 5 marks)

You are given three small sample test cases below. Please find the assignments that result in the minimum **max-load** for each test case.

1. We have  $m = 2$  processors and  $n = 6$  tasks with length  $\{1, 4, 3, 12, 13, 1\}$  (1 mark).
2. We have  $m = 3$  processors and  $n = 7$  tasks with length  $\{5, 7, 2, 1, 2, 1, 2\}$  (2 marks).
3. We have  $m = 4$  processors and  $n = 8$  tasks with length  $\{6, 5, 5, 3, 2, 1, 1, 1\}$  (2 marks).

### B.2 NP-hardness Proof (5 marks)

Prove the NP-hardness of this optimization problem: MULTIPROCESSOR-SCHEDULING by reducing an NP-hard problem PARTITION (see Subsection A.5) into an instance of MULTIPROCESSOR-SCHEDULING with  $m = 2$ . PS: There is a more rigorous proof but it is not needed for this subsection.

Hint: Look at sample test case 1 above to help you do this.

### B.3 Greedy Algorithm Part 1 (8 marks)

Now consider the following greedy algorithm:

- Consider the tasks in **any** order.
- Assign a task  $t_i$  to the processor  $p_j$  that currently has the minimum load.

That is, when assigning task  $t_i$ , we calculate the current load on all  $m$  processors and choose the processor  $p_j$  with the smallest load; then we assign  $t_i$  to that processor  $p_j$ . Now show that this greedy algorithm is a 2-approximation of the optimal (i.e., the minimum **max-load**).

*Hint:* Consider the processor with the maximum load and the last task assigned to that processor.

**B.4 Special Case (2 marks)**

If we process the tasks in **any** order as in Subsection B.3, what do you think will happen if  $n \leq m$ ?

**B.5 Greedy Algorithm Part 2 (7 marks)**

Now, consider an alternate greedy algorithm where the tasks are not processed in **any** order, but in **non-increasing** order (we can do this with an initial  $O(n \log n)$  sorting of the  $n$  tasks). Now show that this alternate greedy algorithm is a 1.5-approximation of the optimal (i.e., the minimum **max-load**).

*Hint:* If we process the tasks in **non-increasing** order and  $n > m$  — think about the  $(m + 1)$ -th task added in this scenario.

**B.6 Create Test Case (3 marks)**

The alternate greedy algorithm in Subsection B.5 happen to get optimal answers for all three sample test cases in Subsection B.1. Create a small test case with any  $m \geq 2$  processors and  $n > m$  tasks with lengths of your choosing so that the alternate greedy algorithm in Subsection B.5 does not get an optimal answer (but should be within 1.5 of **max-load**). Explain your test case.

## C M(W)IS on Grid Graph — the Next Level(s) (35 marks)

In Tutorial 03, we have learned about the MAX-INDEPENDENT-SET (MIS) problem: Given a graph  $G = (V, E)$ , pick the maximum-size set  $I \subset V$  so that no two vertices in  $I$  share an edge. You are told that MIS is also an NP-hard optimization problem. In Tutorial 03, we discussed a ‘toy’ approximation algorithm on a graph that is arranged as an unweighted square grid of size  $n \times n$  and the vertices are labeled top-to-down and left-to-right using 1-based indexing.

In this question, we will consider two variants of this problem. First, a variant where the grid can be rectangular of size  $n \times m$  and contain  $k$  holes ( $0 \leq k \leq n \times m$ ), as in Figure 1 (notice the vertex numbering, especially involving the holes):

### C.1 Sample Test Case (2 marks)

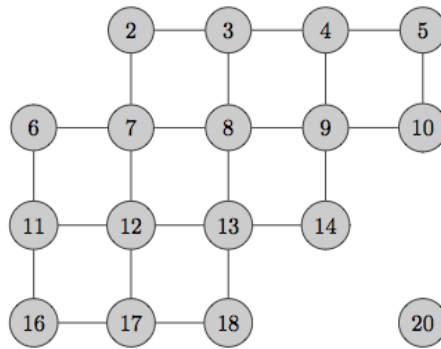


Figure 1: A Grid Graph of size  $4 \times 5$  with 3 holes (vertex 1, 15, and 19 are gone), so  $|V| = 17$  vertices.

What is the MIS (and its size  $|\text{MIS}|$ ) of Figure 1 above?

### C.2 Greedy Algorithm Part 1 (6 marks)

Consider the following greedy algorithm for graph  $G = (V, E)$  that we have discussed in Tutorial 03:

- Set  $I = \emptyset$ ;
- Repeat until  $V$  is empty:
  - Choose **any arbitrary** vertex  $u \in V$ ,
  - Add  $u$  to  $I$ ,
  - Delete  $u$  and all the neighbors of  $u \in V$ .

Is this ‘toy’ greedy algorithm that we discussed in Tutorial 03 still a 2.5-approximation of the optimal MIS solution on the **rectangular** grid graph **with some holes** like in Figure 1 above? If yes, argue why it is still a 2.5-approximation. If no, give a counter example and then provide the correct approximation bound.

### C.3 Max-Weight-Independent-Set on Grid Graph (6 marks)

Now, consider the **weighted** variant: You are also given rectangular grid of size  $n \times m$  and contain  $k$  holes but now with **positive** weight  $w(u)$  for each vertex  $u \in V$ . Our task now is to find the MAX-WEIGHT-INDEPENDENT-SET (MWIS). First, draw a test case (a small rectangular grid graph with at least one hole but this time the vertices are weighted; you are allowed to assign any **positive** vertex weight as you see fit) and argue that the ‘toy’ greedy approximation algorithm from Subsection C.2 above does not work, i.e., it does not have a proven approximation ratio.

### C.4 Greedy Algorithm Part 2 (6 marks)

Now, consider the following ‘upgraded’ greedy algorithm part 2:

- Set  $I = \emptyset$ ;
- Repeat until  $V$  is empty:
  - Choose a vertex  $u \in V$  **with the maximum weight** // compare this with C.2,
  - Add  $u$  to  $I$ ,
  - Delete  $u$  and all the neighbors of  $u \in V$ .

Now argue that this algorithm is a 4-approximation of the optimal MWIS solution on the rectangular grid graph with some holes like in the unweighted (all vertex weights are 1) Figure 1 and the test case that you draw in Subsection C.3 above.

### C.5 Optimal and ‘Universal’ Part 1 (8 marks)

An ex senior CS4234 student remarked to Steven that the ‘toy’ greedy approximation algorithm in Subsection C.2 (for MIS - the unweighted one) is useless. This student claims to be able to **optimally** solve **any** instance of **MIS** on  $n \times m$  rectangular grid graph with  $k$  holes in polynomial time when all vertices have unit weight 1.

If you think that you also support this claim, provide that polynomial time algorithm that can both show the optimal MIS size and the actual vertices selected in the MIS, and analyze its polynomial time complexity.

If you think that this senior student’s remarks is wrong, argue that MIS is actually still an NP-hard optimization problem on such  $n \times m$  rectangular grid graph with  $k$  holes.

### C.6 Optimal and ‘Universal’ Part 2 (7 marks)

Yet another ex senior CS4234 student remarked to Steven that both the ‘toy’ greedy approximation algorithms in Subsection C.2 **and** Subsection C.4 (for MIS and then MWIS) are useless. This other student claims to be able to **optimally** solve **any** instance of MWIS - the weighted one (so that means the solution will also works for MIS where all vertex weight is set to 1) on  $n \times m$  rectangular grid graph with  $k$  holes in polynomial time and each vertex has its own individual **positive** weight...

If you think that you also support this claim, provide that polynomial time algorithm that can both show the optimal MWIS size and the actual vertices selected in the MWIS, and analyze its polynomial time complexity.

If you think that this senior student's remarks is wrong, argue that MWIS is still an NP-hard optimization problem on such  $n \times m$  rectangular grid graph with  $k$  holes.

– End of this Paper, All the Best –



## D Answer Sheets

Write your Student Number in the box below:

A	0							
---	---	--	--	--	--	--	--	--

This portion is for examiner's use only

Section	Maximum Marks	Your Marks	Remarks
A	35		
B	30		
C	35		
Total	100		

Box A-1.

--

Box A-2.

--

Box A-3.

--

Box A-4.

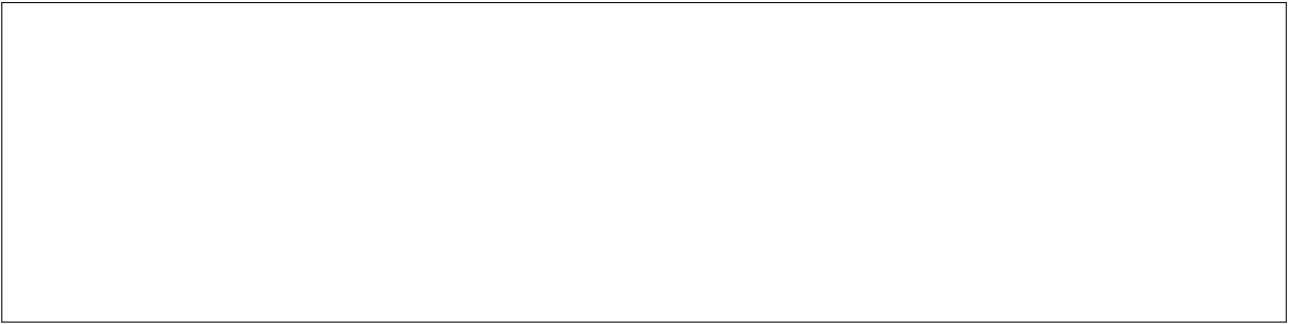
Box A-5.

Box A-6 a+b.

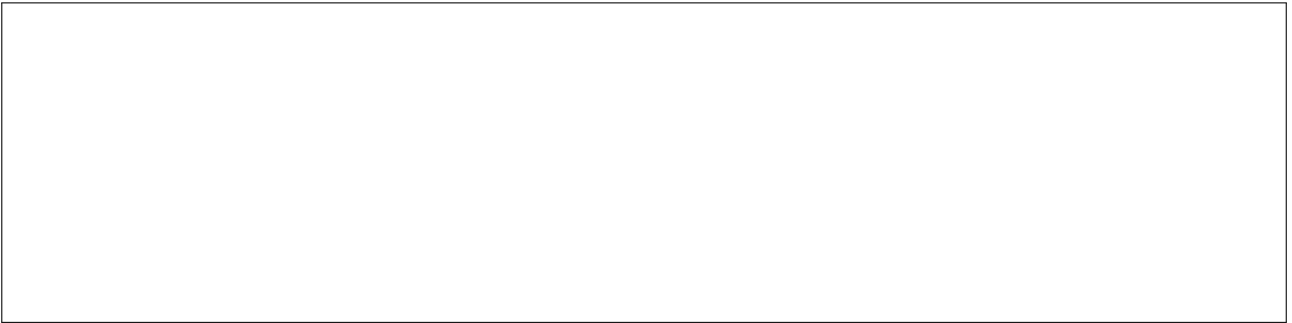
Box A-7.

Section A Marks =  +  +  +  +  +  +  =

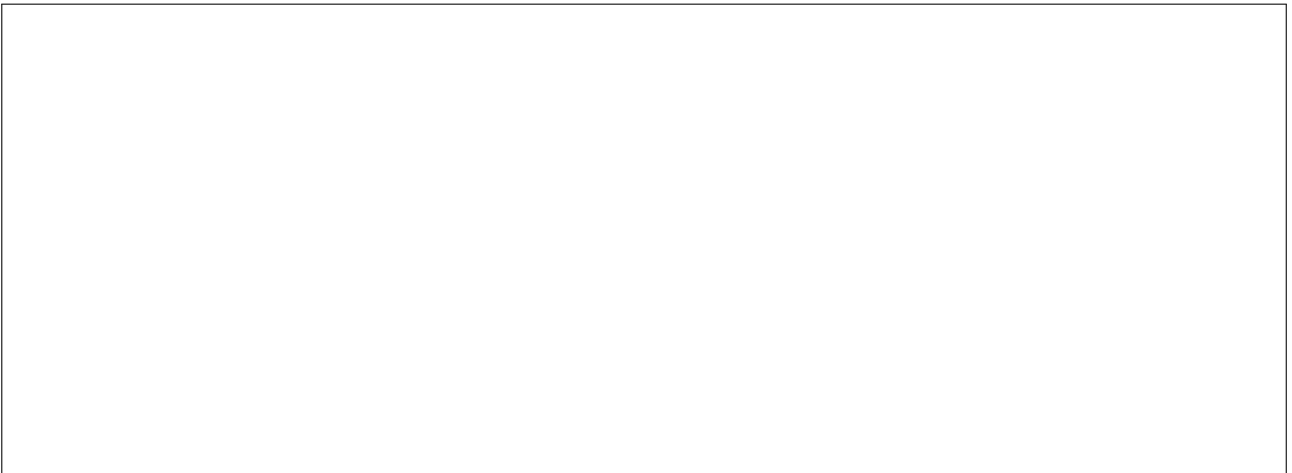
Box B-1.



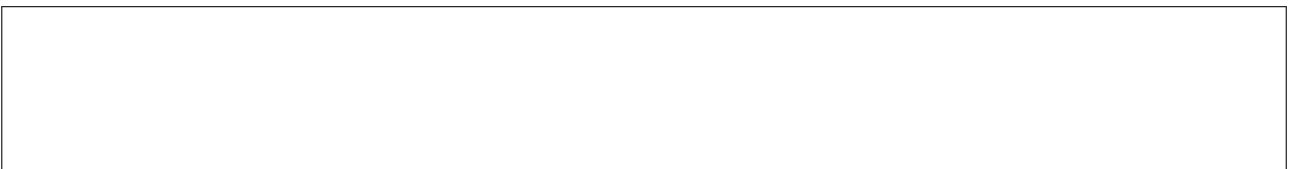
Box B-2.



Box B-3.



Box B-4.



Box B-5.

Box B-6.

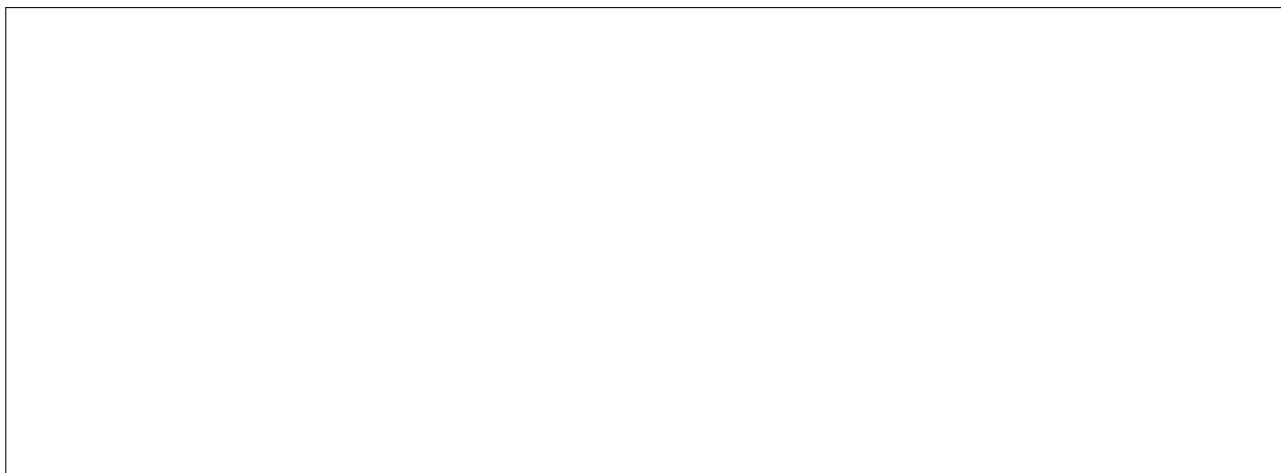
Section B Marks = \_\_ + \_\_ + \_\_ + \_\_ + \_\_ + \_\_ = \_\_\_

Box C-1.

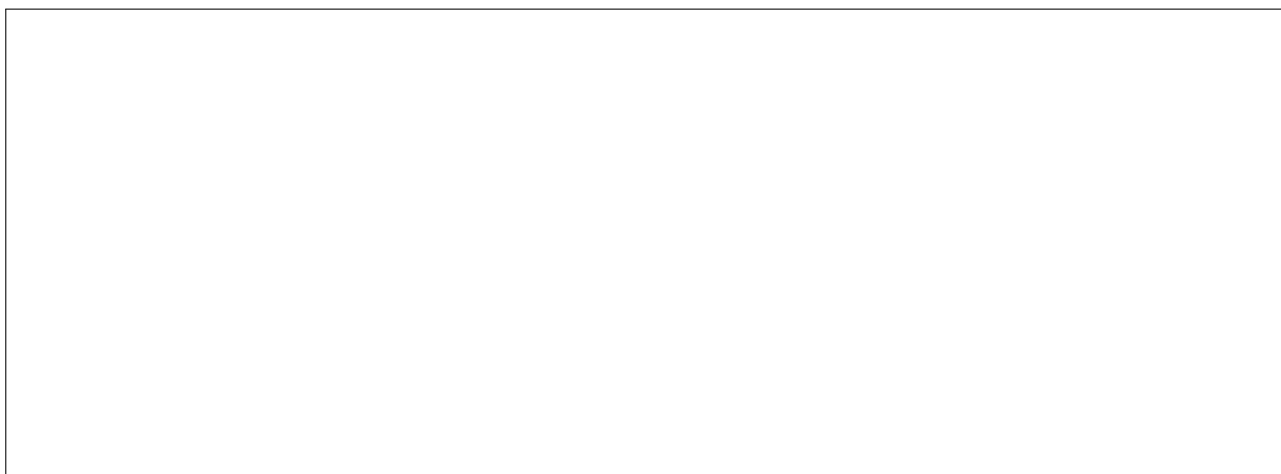
Box C-2.

Box C-3.

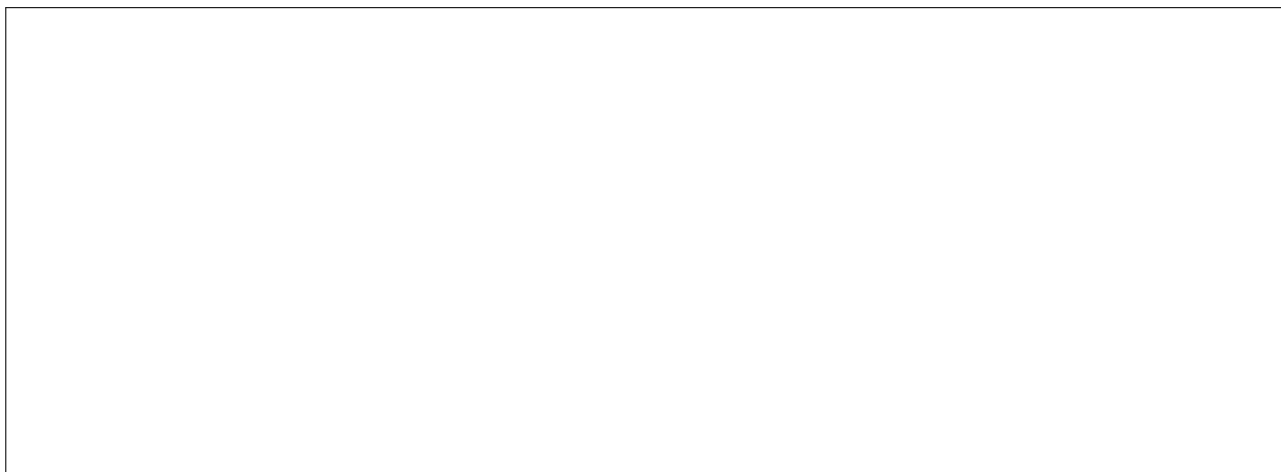
Box C-4.



Box C-5.



Box C-6.



Section C Marks =  +  +  +  +  +  =

End of this Paper, All the Best, You can use this Page 14 for extra writing space.