SMU
SINGAPORE MANAGEMENT UNIVERSITY

School of Information Systems

Past Year Paper: AY 2012-13 Term 2 Final Examinations with Solutions

Date and Start Time	23 April 2013 1-4pm
Course	**IS103 – Computational Thinking**
Group	
Instructor	
Matriculation Number	

INSTRUCTIONS TO CANDIDATES

1. PLEASE DO NOT TURN OVER UNTIL TOLD TO DO SO.

2. The time allowed for this examination paper is **3 hours**.

3 This examination paper contains a total of **5** questions and comprises **18** pages including this instruction sheet.

4 You are required to answer **ALL** questions.

5 **Write all answers on this paper.** You may use the back pages if necessary.

6 Note that most questions require short answers. Long-winded explanation will be penalized.

Question	Marks	Awarded Marks
1. MCQ	10	
2. Stacks/Queues	9	
3. Trees	9	
4. Graphs	10	
5. Heuristics	12	
Total	50	

Question 1: Multiple-Choice Questions (10 marks)

1.1 This examination paper contains 5 questions, and you need to answer all questions. But suppose you only need to answer any 4 questions. In how many different ways can you attempt this paper?

(a) 5!
(b) $^5P_4$
(c) $^5C_4$
(d) 4!
(e) 5 x 4

Answer: (c)

1.2 Algorithm analysis should be independent of all of the following ***except*** ______.

(a) the programming language used in the implementation of the algorithm
(b) the computer used to run a program which implements an algorithm
(c) the number of operations in an algorithm
(d) the test data used to test a program which implements an algorithm
(e) the experience of the person implementing the algorithm

Answer: (c)

1.3 Which of the following is used to compare the computational efficiency of two algorithms?

(a) their growth rate functions
(b) their number of lines of code
(c) the programming languages they are written in
(d) the programmers who wrote them
(e) the data structures they used

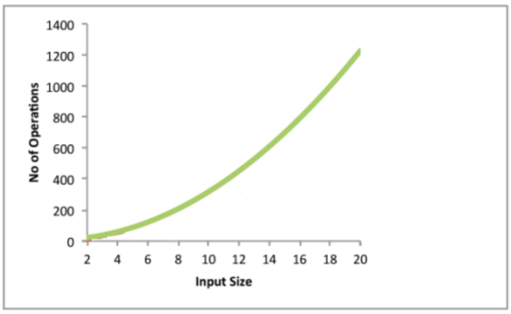
Answer: (a)

1.4 If a problem of size n requires a computational time of at most k^2 x n units, where k is a constant, then the problem is said to run in ______ time.

(a) O(1)
(b) O(n)
(c) $O(k^2n)$
(d) O(kn)
(e) $O(k^2)$

Answer: (b)

1.5 A particular algorithm takes in a variable input size (number of input elements) and performs a certain number of operations to compute the result. The following graph shows the relationship between the number of required operations and the input size.



What can you infer about this algorithm?

(a) It is an inefficient algorithm.
(b) Its Big-O complexity is O(n^2).
(c) This algorithm performs an exhaustive search.
(d) It uses recursion.
(e) The algorithm is an insertion sort.

Answer: (b)

1.6 You are given a graph with n vertices and m edges represented by an adjacency list. Suppose you have an algorithm that reasons on this graph, and contains two separate loops – one loops through all the vertices, and the other loops through all the edges of the graph. What can you say about the worst case time complexity of this algorithm?

(a) O(n)
(b) O(m)
(c) O(nm)
(d) O(n+m)
(e) O(n log m)

Answer: (d)

1.9 Suppose we know a problem X is in the class P, and Y is NP-complete.

Which of the following statements are true?

1. X is in NP.
2. Y is in NP.
3. X is NP-complete.
4. X can be reduced to Y.
5. Y can be reduced to X.

(a) 1 and 2 only.
(b) 2 and 3 only.
(c) 1, 2 and 3 only.
(d) 1, 2 and 4 only.
(e) 1, 2, and 5 only.

Answer: (d)

1.10 Suppose someone comes to you with an NP-hard optimization problem. Which of the following advice reflects computational thinking?

1. Solve it exactly using backtracking search
2. Solve it heuristically using a greedy algorithm
3. Implement the solution in Python instead of Java
4. Reduce/map it to another NP-hard optimization problem
5. Reduce another NP-hard optimization problem to it

(a) 1 and 2 only.
(b) 2 and 3 only.
(c) 1, 2 and 3 only.
(d) 1, 2 and 4 only.
(e) 1, 2, and 5 only.

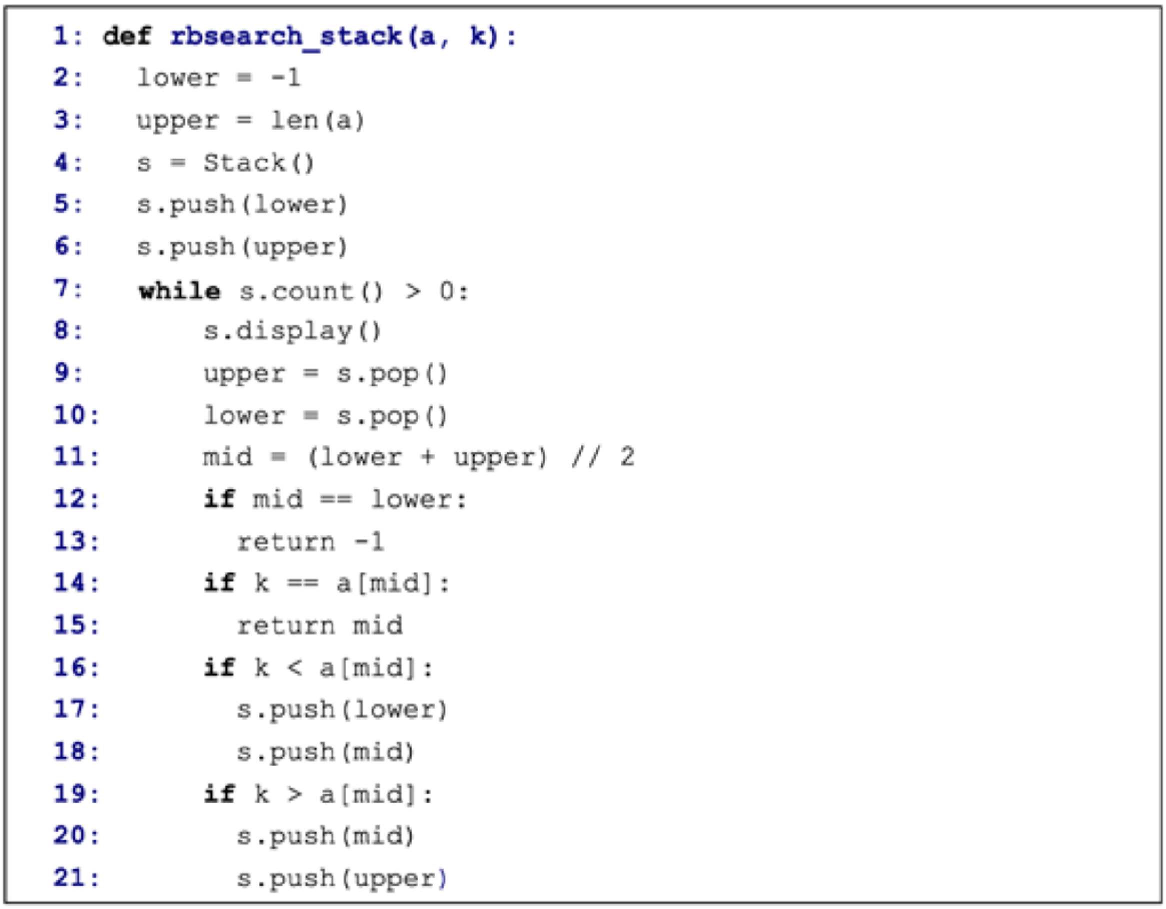
Controversial. This question hasn’t got the best options, but if we had to choose the "best" answer, then: answer: (d).

Because:

- (1) is OK with a small n value
- (4) is OK if you already have a heuristic solution to another problem that you can map the current one to.

Question 2. Stacks and Queues (9 marks)

Binary search is an algorithm to search an array based on the principle of decomposition. The following is a stack-based version of binary search algorithm rbsearch_stack, which returns as output the index position where the search key k can be found in the input array a (or -1 if the key is not found).



```
1: def rbsearch_stack(a, k):
2:     lower = -1
3:     upper = len(a)
4:     s = Stack()
5:     s.push(lower)
6:     s.push(upper)
7:     while s.count() > 0:
8:         s.display()
9:         upper = s.pop()
10:        lower = s.pop()
11:        mid = (lower + upper) // 2
12:        if mid == lower:
13:          return -1
14:        if k == a[mid]:
15:          return mid
16:        if k < a[mid]:
17:          s.push(lower)
18:          s.push(mid)
19:        if k > a[mid]:
20:          s.push(mid)
21:          s.push(upper)
```

Figure 1 Stack-based Algorithm for Binary Search

2(a) (3 marks) Line 8 in Figure 1 prints out the content of the stack during each execution of the while loop. Suppose that we call rbsearch_stack(a, k) with the following inputs:

a = [2, 19, 27, 31, 45, 62, 67, 74, 92]
k = 74

Show the contents of the stack after **every call** to line 8 for these inputs.

Answer:

Iteration	Content of Stack (top to bottom)
1	9 (top), -1
2	9 (top), 4
3	9 (top), 6

2(b) (3 marks) A queue can be used to simulate customers waiting in a single queue in front of single teller (server). You are given the following information on the arrival time and the transaction duration of each customer (time starts at 0):

Customer	Arrival time	Transaction Duration
C1	5	5
C2	7	9
C3	8	4
C4	23	6
C5	30	5
C6	33	4

Show the content of the queue at the various time points in the table below (note that the customer being served stays at the front of the queue until the end time of her transaction):

Time	Content of Queue (front to back)
0	Empty
5	C1
7	C1 C2
8	C1 C2 C3
10	C2 C3
19	C3
23	C4
29	Empty
30	C5
33	C5 C6
35	C6
39	Empty

2(c) (3 marks) Now suppose the bank wants to maintain a **priority queue** where a customer with the *highest* priority (i.e. smallest value, 1 being the highest priority, 2 is next in priority, etc) will always be served whenever the server becomes available. Customers with the same priority will be served according to their time of arrivals (i.e. first-come-first-served). We further assume no pre-emption, i.e. when a customer is being served and another high-priority customer arrives, the latter will have to wait until the former completes his transaction.

Now given the following information on the arrival time, customer priority, and the transaction duration of each customer (time starts at 0):

Customer	Arrival time	Priority	Transaction Duration
C1	5	3	5
C2	7	2	9
C3	8	1	4
C4	22	3	6
C5	23	1	5
C6	33	2	4

Show the content of the **priority queue** at the various time points in the table below (note that the customer being served stays at the front of the queue until the end time of her transaction):

Time	Content of Queue (front to back)
0	Empty
5 C1 arrives. C1 starts.	C1(3)
7 C2 arrives.	C1(processing). C2 (2)
8 C3 arrives. C3 higher priority than C2.	C1(processing). C3(1) C2(2).
10 C1 ends. C3 starts.	C3(processing). C2(2)
14 C3 ends. C2 starts	C2(processing)
19	C2(processing)
22 C4 arrives.	C2(processing), C4(3)
23 C5 arrives. C5 higher priority than C4. C2 ends.	C5(processing), C4(3)
28 C5 ends. C4 starts	C4(processing)
29	C4
30	C4
33 C6 arrives.	C4(processing). C6(2)
34 C4 ends. C6 starts	C6 (processing)
35	C6 (processing)
38 C6 ends	Empty
39	Empty

Inserted rows in red (not part of answer). Priority behind the element in brackets also not required in answer.

Question 3. Tree and Recursion (9 marks)

In a planet X, each inhabitant can reproduce at most 2 children. Hence, a family tree can be represented by a binary tree. An example of such a family tree is given in Figure 2.

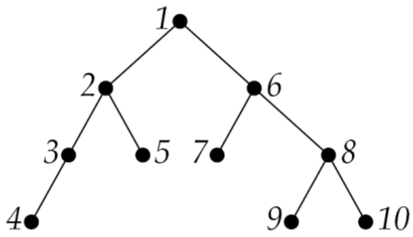


Figure 2

Assume that each tree node t has four attributes t.key, t.leftChild, t.rightChild and t.descendents that store the key, left child pointer, right child pointer, and number of descendents respectively.

For example, for the binary tree shown in Figure 2, there are 4 inhabitants who have exactly two children (namely, nodes 1, 2, 6 and 8); and the number of descendents for nodes 1 to 10 are 9, 3, 1, 0, 0, 4, 0, 2, 0 and 0 respectively.

Given a family tree rooted at t,

3(a) (3 marks) Design a recursive divide-and-conquer algorithm CountTwoChildren(t) that returns the number of inhabitants who have *exactly* two children.

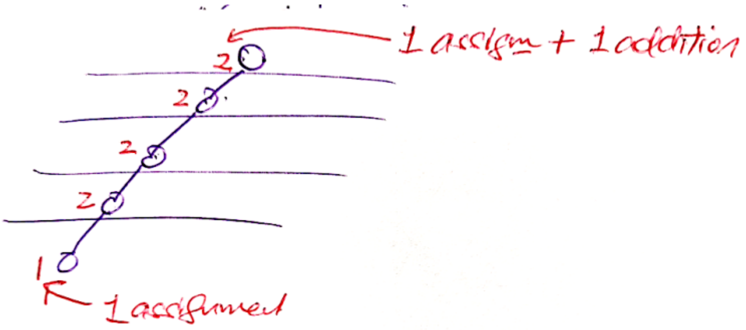
```
def CountTwoChildren(t)
  # leaf node. No child
  if t.leftChild == None and t.rightChild == None
    return 0
  # only left child
  if t.leftChild != None and t.rightchild == None
    return CountTwoChildren(t.leftChild)
  # only right child
  if t.leftChild == None and t.rightchild != None
    return CountTwoChildren(t.rightChild)
  # 2 children
  return 1 + CountTwoChildren(t.leftChild) + CountTwoChildren(t.rightChild)
```

(4 marks) Design a <u>recursive</u> divide-and-conquer algorithm CountDescendents(t) that sets the number of descendents for each inhabitant in the tree. That is, you are required to set the values of the descendents attribute, for all the nodes in the tree.

```
# Need to set each node’s t value as well.
def CountDescendents(t):
  if t.leftChild == None and t.rightChild == None:
    t.descendents = 0
    return 0
  elsif t.leftChild == None and t.rightChild != None:
    t.descendents = 1 + countDescendents(t.rightChild)
    return t.descendents
  elsif t.leftChild != None and t.rightChild == None:
    t.descendents = 1 + countDescendents(t.leftChild)
    return t.descendents
  else  # have 2 children
    t.descendents = 2 + countDescendents(t.leftChild) +
countDescendents(t.rightChild)
    return t.descendents
```

3(b) (2 marks) What is the worst-case time complexity of your algorithms? Explain.

Consider a left-skewed tree like this:



There are 2 operations (1 assignment & 1 addition) when there is only 1 child (according to the algo above). And 1 operation (1 assignment only) when there is no child.

From the diagram →
h (height of tree) = n (number of nodes)
no. of ops = 2 + 2 + 2 + 2 +…. + 1
= 2(n-1) + 1 ➔ O(n), where n is the number of nodes

Try the same reasoning with a full binary tree; you should get the same time complexity.

Question 4. Graph and Topological Sort (10 marks)

4(a) (2 marks) In no more than 30 words, explain the difference between a topological sort and an ordinary sort (such as insertion or quick sort).

There could be >=1 "correct" ways to perform a topo sort for a DAG, but only 1 correct way to perform an ordinary sort.

Refer to the following directed acyclic graph (DAG):

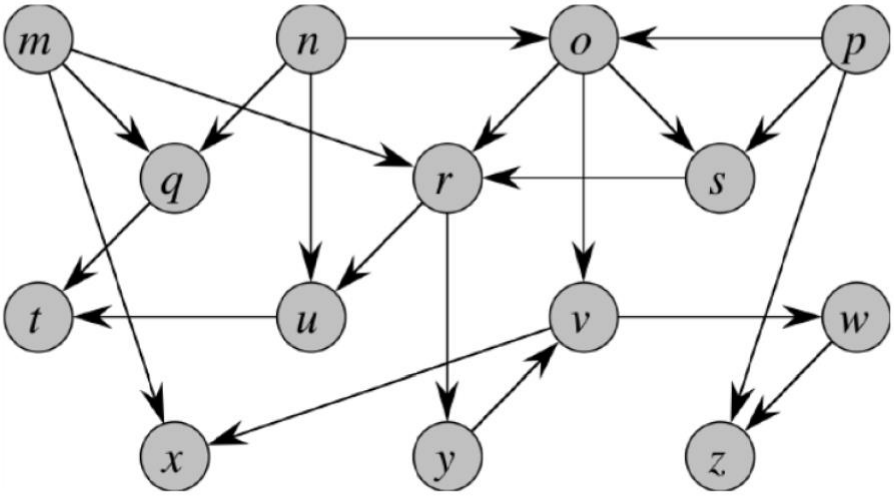


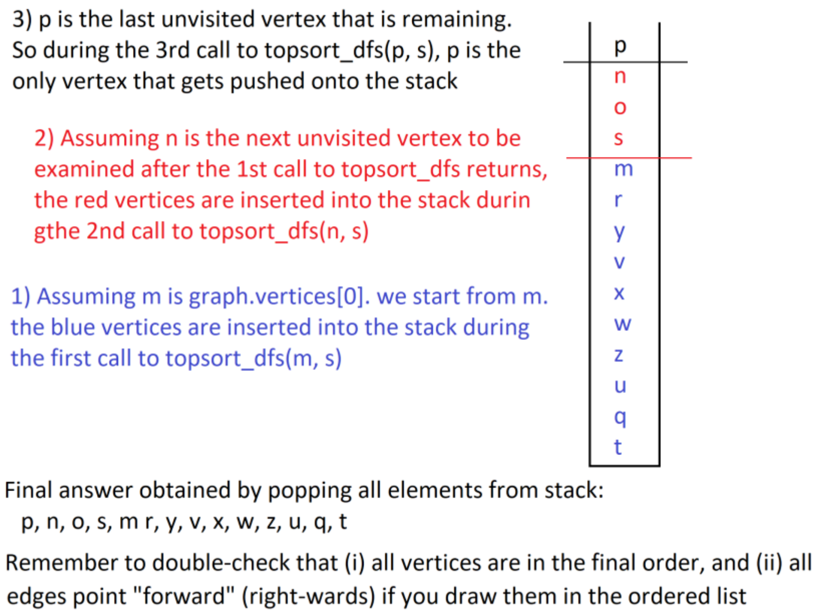
Figure 3

4(b) (3 marks) Give a topological ordering of the DAG in Figure 3. You should also show all the edges between the nodes along the sequence.

Answer:

By using the topo algo given in your slides, and starting with vertex "m", we will get:

p, n, o, s, m, r, y, v, x, w, z, u, q, t



3) p is the last unvisited vertex that is remaining. So during the 3rd call to topsort_dfs(p, s), p is the only vertex that gets pushed onto the stack

2) Assuming n is the next unvisited vertex to be examined after the 1st call to topsort_dfs returns, the red vertices are inserted into the stack durin gthe 2nd call to topsort_dfs(n, s)

1) Assuming m is graph.vertices[0]. we start from m. the blue vertices are inserted into the stack during the first call to topsort_dfs(m, s)

Final answer obtained by popping all elements from stack:
p, n, o, s, m r, y, v, x, w, z, u, q, t

Remember to double-check that (i) all vertices are in the final order, and (ii) all edges point "forward" (right-wards) if you draw them in the ordered list

Other correct solutions (by observation) are also accepted. e.g. :

m n p o q s r u y v t w z x

There is another easy algorithm called the Kahn algorithm that can be used to identify all topo orders. You can check it out online. You may prefer to use Kahn's algo to identify one correct topo order instead. This particular question does not specify the algo that you should use to come up with your answer.

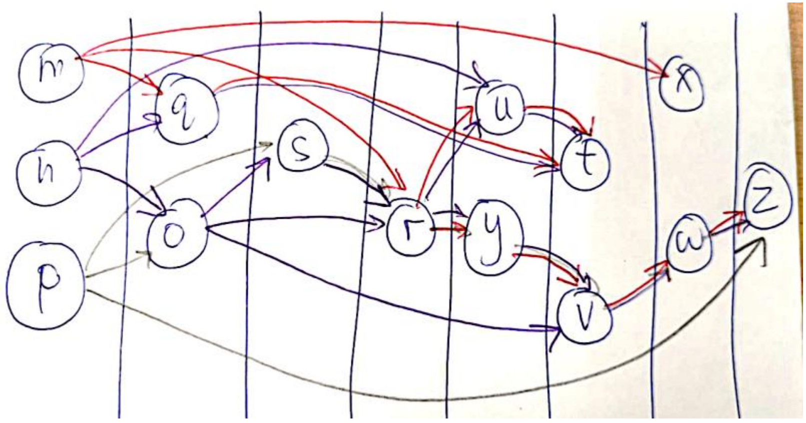
4(c) (3 marks) Suppose a DAG is used to model a project network, where the nodes represent the tasks and edges represent precedence relationships. In the example above, you can start task *q* only upon completion of tasks *m* and *n*. Suppose you have unlimited resources and can execute as many tasks as you can at the same time, and suppose each task takes exactly 1 month to complete.

Discuss how you may make use of the topological ordering to determine the least number of months needed to complete the entire project. Using your approach, how long does it take to complete the project given in Figure 3?

Answer:

There are 14 nodes (tasks). The 3 tasks with no dependencies are p, n and m. They should start concurrently during the 1st month.

I mapped the nodes into different tiers (or columns, in this diagram) using Kahn's algorithm. So, m, n and p are the nodes with zero in-degree. Once they are removed, q and o are the nodes with zero in-degree. Once q and o are removed, s is the only node with zero in-degree… etc. This is the resultant diagram:



Look for the "longest chain". It should be n, o, s, r, y, v, w, z. Even if n starts on the 1st month, this chain requires 8 months to complete.

4d. (2 marks) Now suppose that due to a manpower crunch, you have limited resource capacity such that not all tasks can be executed concurrently. (As an example, think of having 3 tasks that can be executed concurrently, each requiring 1 man to perform, but only 2 workers are available. And workers cannot multi-task. So one of the tasks will need to be put hold until a resource becomes available.) In less than 50 words, discuss why this problem might become computationally intractable.

Answer:

The problem here is: use topo sort to determine the shortest time to complete all the tasks, given n tasks (nodes) and w workers.
From the diagram in the previous answer, you can come up with all the possible topo orders for the graph using Kahn's algorithm. There can be a large number of possible topo orders depending on the number of nodes and how they are connected.

For each valid topo order:

- Select the first w tasks in the current order to perform concurrently
- If that is impossible (e.g. if one of the two tasks needs the other to be completed first), then select the next node instead if possible
- Compute the total about of time required to complete all the tasks

This becomes a combinatorial problem because there can be many possible combinations of valid topo orders, and the large number of possible combinations that need to be tested may make this problem intractable.

Question 5. Greedy Algorithm (12 marks)

You have just completed the IS103 Computational Thinking course, and desperately need a holiday break. You get together with a few good friends, and agree to have a road trip from Singapore (S) to 4 cities in Malaysia, namely Kuala Lumpur (K), Melaka (M), Ipoh (I), and Penang (P), and finally return back to Singapore again.

To save money on accommodation and petrol, you plan to visit each Malaysian city exactly once, and like to find the shortest possible tour.

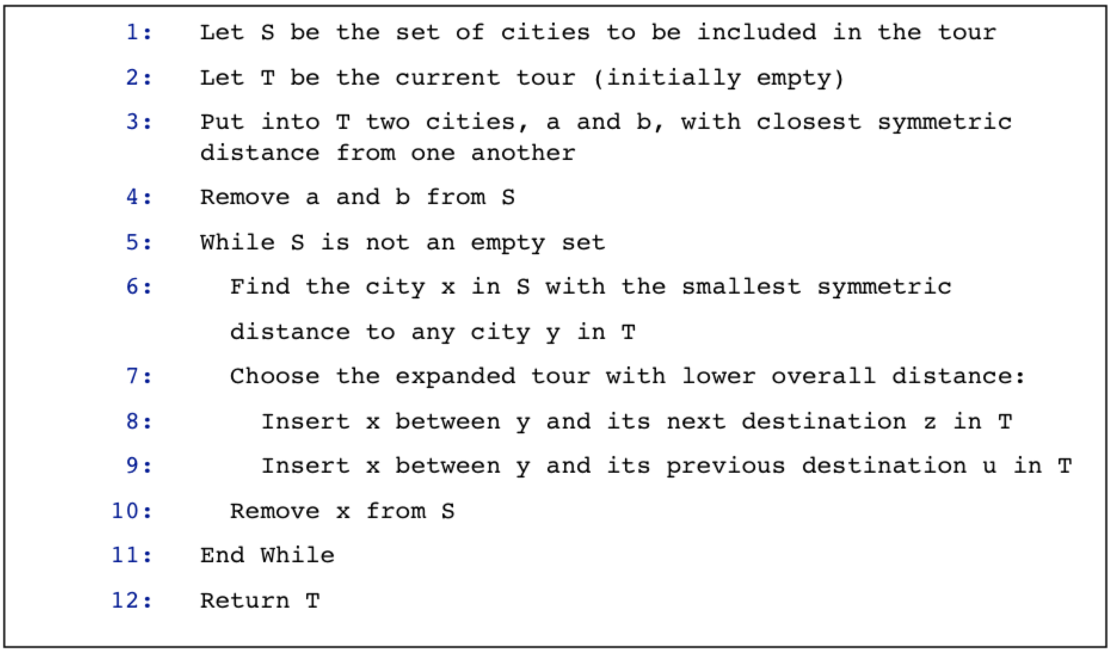
The table below specifies the distance between any two cities in tens of kilometers.

	I	**K**	**M**	**P**	**S**
I	-	20	36	12	55
K	20	-	15	31	35
M	36	15	-	46	22
P	12	31	46	-	70
S	55	35	22	70	-

5(a) (1 mark) What is the **total number of possible tours starting and ending in Singapore**?

Answer:
4!/2

5(b) (4 marks) The pseudo-code for the greedy algorithm for the Traveling Salesman problem is given below.



```
Let S be the set of cities to be included in the tour
Let T be the current tour (initially empty)
Put into T two cities, a and b, with closest symmetric
distance from one another
Remove a and b from S
While S is not an empty set
  Find the city x in S with the smallest symmetric
  distance to any city y in T
  Choose the expanded tour with lower overall distance:
    Insert x between y and its next destination z in T
    Insert x between y and its previous destination u in T
  Remove x from S
End While
Return T
```

What is the **tour resulting from the greedy algorithm** for your trip? Show your working clearly.

Answer:

Step 1: choose I-P
Step 2: choose K (because K-I is 20) → IKP (this means I-K-P and back to I)
Step 3: choose M (because M-K is 15).
Choose between IMKP or IKMP
Choose IKMP (shorter route) → IKMP
Step 4: Choose S (S-M is 22)
Choose between IKSMP or IKMSP
Choose IKSMP (shorter route) → IKSMP
Final answer: IKSMP, and back to I

Remember that “rotated” answers (e.g. KSMPI, or SMPIK) are identical since we are talking about a cycle.
Also, the “mirror” answer is also identical (e.g. IKSMP is the same as PMSKI)

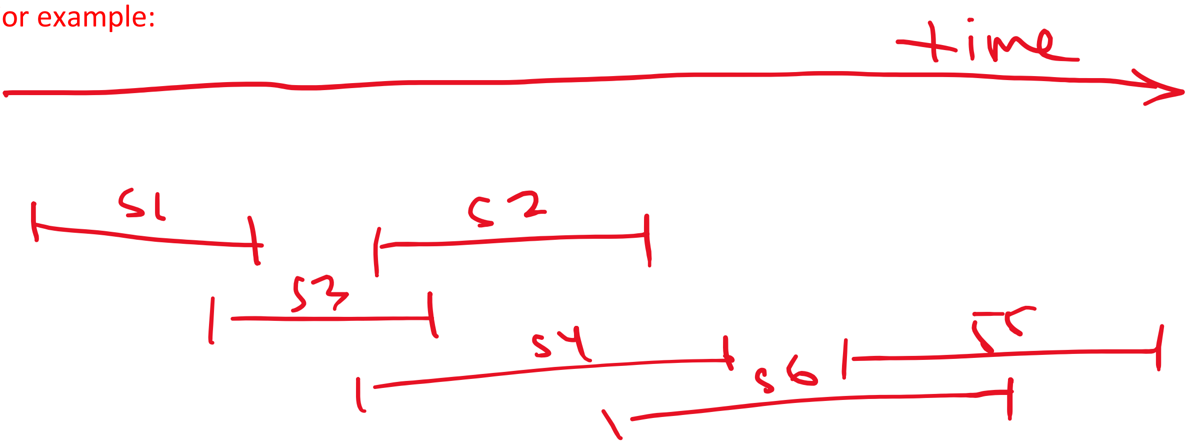
Within a city to be visited, you are also in charge of designing the schedule for visiting the attractions. You ask your friends to submit their requests. Each request specifies the attraction to be visited, starting time, and ending time, e.g., (Petronas Tower, start 9am, end 11am).

5(c) (1 mark) Suppose there are n number of such requests, and there is no duplicate attractions among the requests. How many possible request combinations are there in total to form a schedule? State your answer in terms of n.

Answer:

The requests may overlap in time. So accepting one may prevent us from accepting another.

For example:



If we accept S1, we cannot also accept S3 because of overlapping. So in this case, the schedule with the most number of requests would be 3, for example: {S1, S2, S5}, or {S1, S4, S5}.

However, Q5(c) is a counting question. So, assuming that there is no overlap in all the n requests, we can create 2^n schedules by including or excluding each of the n requests. Answer: 2^n.

Clearly, you cannot accept all requests because some of them conflict with one another. Two requests conflict if they overlap in time. For example:

- (Petronas Tower, start 9am, end 11am) conflicts with (Sunway Lagoon, start 10am, end 11.30am).
- (Petronas Tower, start 9am, end 11am) does ***not*** conflict with (KLCC, start 11am, end 1pm).

For simplicity, you may assume you can zip yourself from one attraction to the next with zero travel time, and there is no queue to get into the attraction. Your task is to design a **schedule that accommodates as many requests as possible without any conflict**.

5(d) (2 marks) For the Knapsack problem, we are given a set of items. Each item has a weight and a value. The objective is to determine which subset of items to be included in a knapsack (a bag) so as to maximize the total value of the included items, while keeping the total weight of the knapsack within a specified limit. In less than 50 words, explain the similarity between this scheduling problem and the **Knapsack problem**.

Answer:

Similarities: combinatorial, subsets, 2^n possibilities
Differences: optimization objective (total value vs. count), constraints (overlap vs. total weight)

5(e) (4 marks) You decide to use the following *greedy algorithm*:

```
1: Let R be the set of submitted requests.
2: Let A be the set of accepted requests, initially empty.
3: While R is not empty
4:     Choose a request r in R based on criterion c
5:     Add r into A
6:     Remove r from R
7:     Remove all other requests that conflict with r from R
8: End While
9: Return A
```

On line 4, there are several options for the greedy criterion c:

- **Option 1**: the request that starts the earliest
- **Option 2**: the request that finishes the earliest
- **Option 3**: the request with the shortest interval (end time - start time)
- **Option 4**: the request with the fewest number of conflicting requests in R

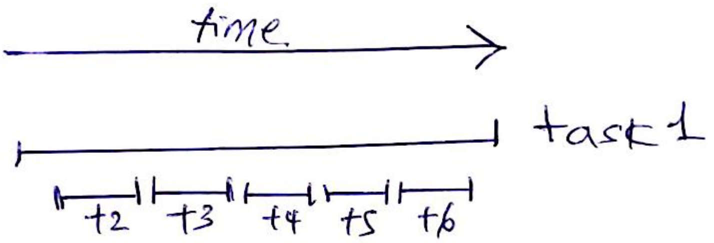
Which one of these four options is the most likely to lead to the **optimal solution** (highest number of accepted requests)? Explain your reasoning.
Hint: You may solve this by elimination. For each option, illustrate a case when that option will fail to produce the optimal solution. If you can identify such cases for three options, then the remaining option is likely to be the optimal one.

Answer:

The optimal answer is Option 2. Students are supposed to produce examples where each option fails. The optimal is circled below.

Example in which OPTION 1 fails (a long request blocks many smaller requests):

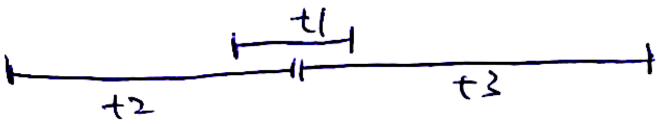
eg in which option 1 fails.



- task 1 would be selected.
- optimal answer : t2 → t6 (5 tasks)

Example in which OPTION 3 fails (a short request blocks two long requests):

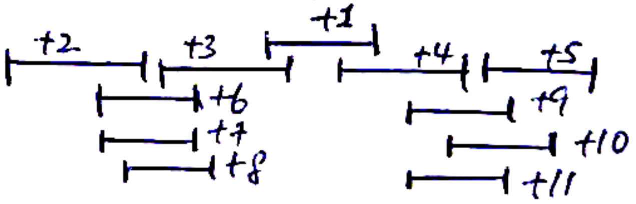
eg in which option 3 fails:



- task 1 selected.
- optimal : t2 & t3

Example in which OPTION 4 fails (a less conflicted request blocks others):

eg in which option 4 fails:



- task 1, 2 & 5 selected.
- optimal: t2, t3, t4, t5

- End of Paper -