Reserved Cells

Ex. 1	
Ex. 2	
Ex. 3	
Ex. 4	
Ex. 5	
Ex. 6	
Tot.	

Algorithms and Programming

22 February 2019

Part I: Theory

Register Number _____ Family Name _____ First Name _____

Course: ◯ 01*OGDLP* 10 credit ◯ 02*OGDLM* 12 credit

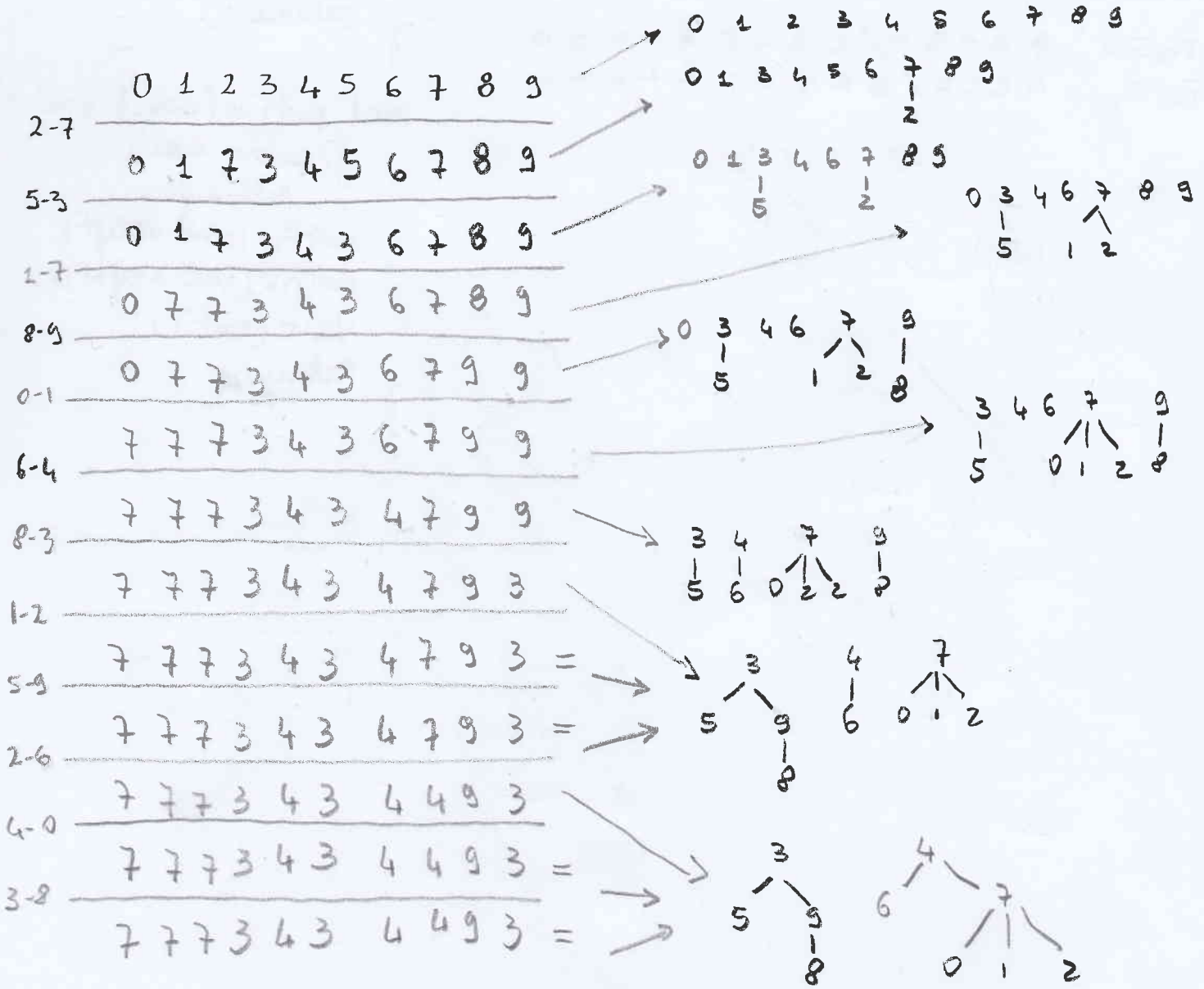
No books or notes are allowed. Solve exercises directly within the reserved space. Additional sheets are accepted only when strictly necessary. Available time: 60 **minutes.**

1. **(2.0 points)**
   Given the following sequence of pairs, where the relation i-j means that node i is adjacent to node j:

   2-7 5-3 1-7 8-9 0-1 6-4 8-3 1-2 5-9 2-6 4-0 3-8

   apply an on-line connectivity algorithm with quick-union, showing at each step the content of the array and the forest of trees at the final step. Node names are integers in the range from 0 to 9.

   Describe the differences (in terms of logic and complexity) among quick-find, quick-union, and weighted quick-union.



Quick FIND: Find cost constant
Union cost linear
Tot: #pairs · array size

$\forall_{p-q}$ change all id[p] into id[q]

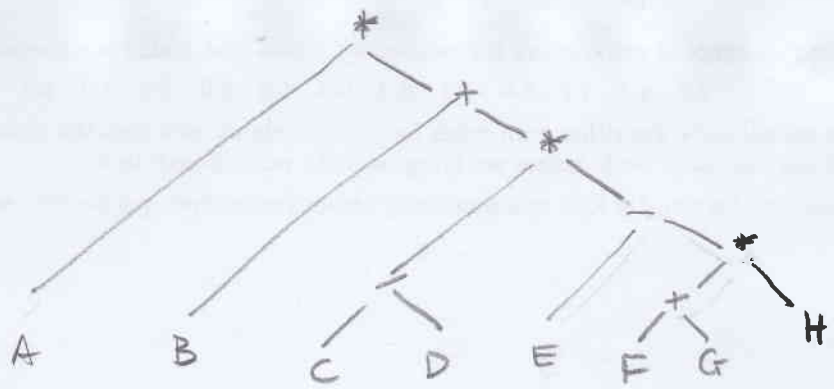
Quick UNION: Find cost linear
Union cost constant
Tot: #pairs · chain length
Follow the chain (id[p])* and change one value at most

Weighted Quick UNION: shorten the chain length of quick-union by putting the smaller tree below the larger one

2. (2.0 **points**)
10 credit course (*01OGDLP*)
Convert the following expression from in-fix to pre-fix notation (Polish Notation) and from in-fix to post-fix notation (Reverse Polish Notation).

$$A * [B + (C/D) * (E - (F + G) * H)]$$

Write two C functions that, given the binary tree used to represent the expression, are able to generate the pre-fix and the post-fix notations.



Pre Fix * A + B * / C D - E + + F G H

Post Fix A B C D / E F G + H * - * + *

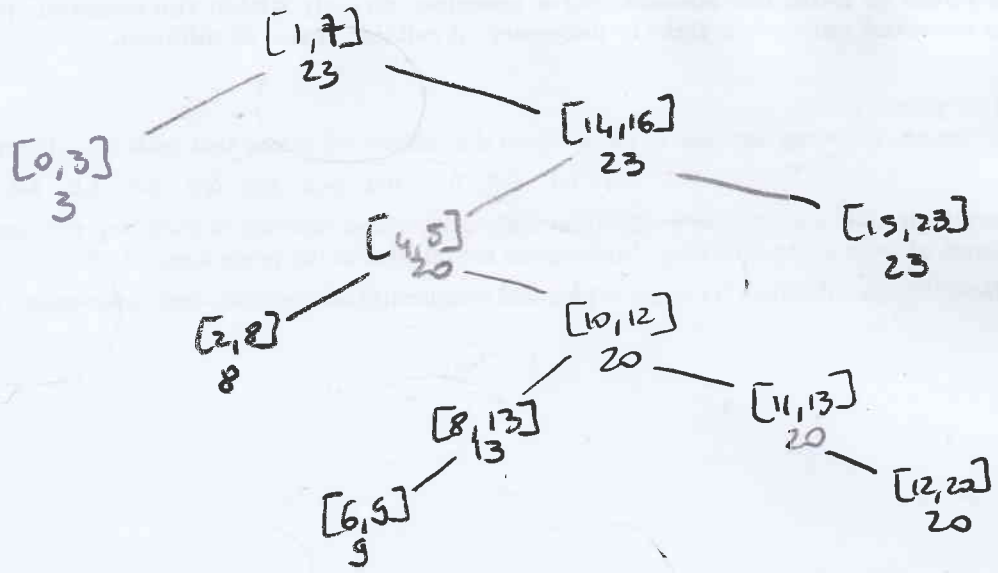
```
void preFix (node_t *root) {
    if (root == NULL)
        return;
    VISIT (root);
    preFix (root -> left);
    preFix (root -> right);
    return;
}
```

```
void postFix (node_t *root) {
    if (root == NULL)
        return;
    postFix (root -> left);
    postFix (root -> right);
    VISIT (root);
    return;
}
```

12 credit course (02*OGDLM*)
Given an initially empty Interval BST, insert in the leaves the following closed intervals:

$[1,7]$ $[14,16]$ $[4,5]$ $[10,12]$ $[8,13]$ $[2,8]$ $[11,13]$ $[12,20]$ $[6,9]$ $[0,3]$ $[15,23]$

After that, search in the resulting Interval BST an intersection with intervals $[12,13]$, and $[19,26]$. Describe all recursion steps performed to search these intervals, and specify when it is necessary to recur on the right child and when to recur on the left one.



Search $[12,13]$:

$[12,13] \cap [1,7]$ NO
$12 > 3$ recur right
$[12,13] \cap [14,16]$ NO
$12 < 20$ recur left
$[12,13] \cap [4,5]$ NO
$12 > 8$ recur right
$[12,13] \cap [10,12]$ YES

Search $[19,26]$

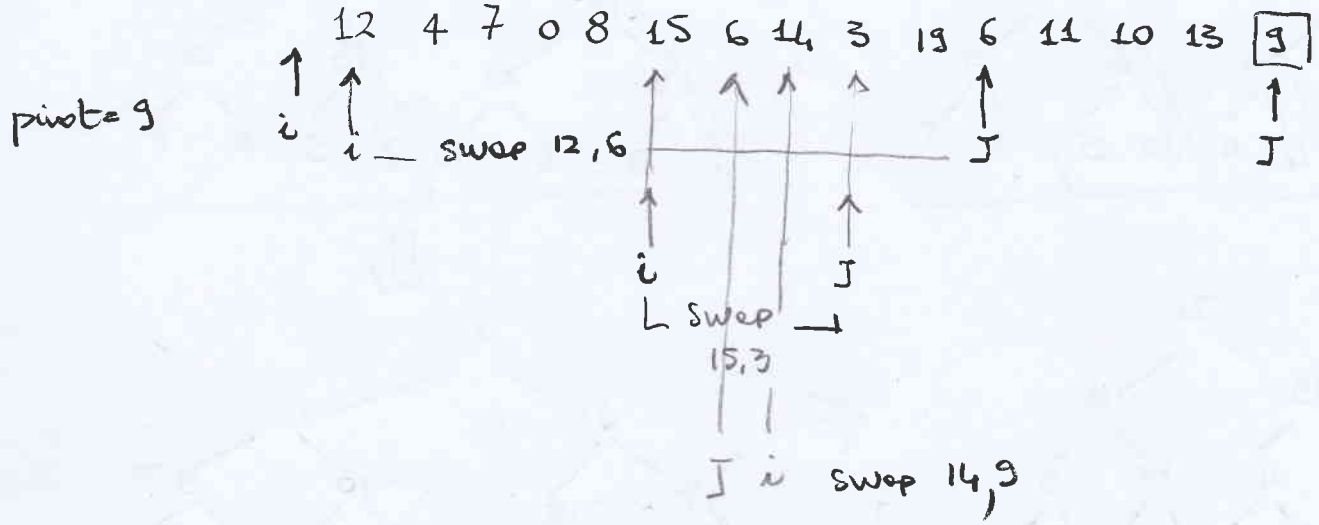
$[19,26] \cap [1,7]$ NO
$19 > 3$ recur right
$[19,26] \cap [14,16]$ NO
$19 < 20$ recur left
$[19,26] \cap [4,5]$ NO
$19 > 8$ recur right
$[19,26] \cap [10,12]$ NO
$19 > 13$ recur right
$[19,26] \cap [11,13]$ NO
recur right
$[19,26] \cap [12,20]$ YES

3. **(2.0 point)**
Given the following sequence of integers stored into an array:

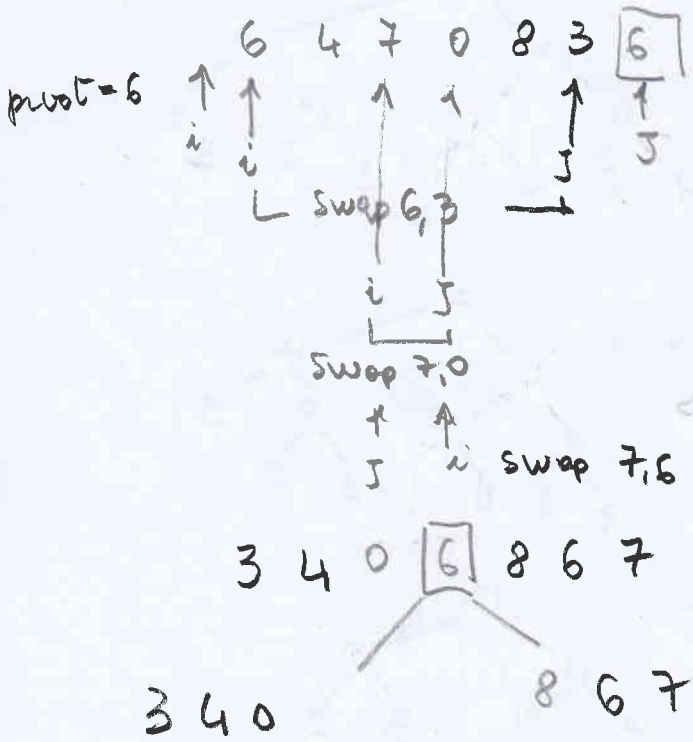
12 4 7 0 8 15 6 14 3 19 6 11 10 13 9

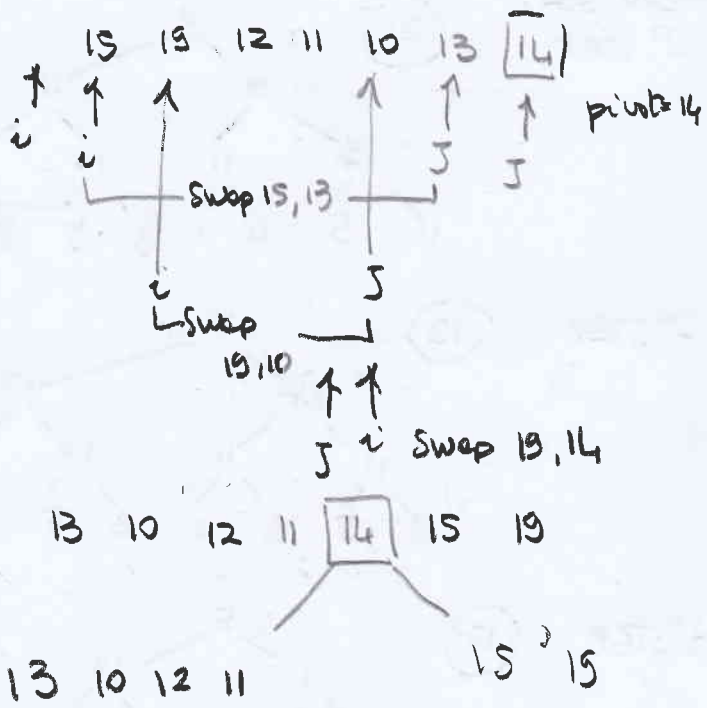
perform the first 2 steps of quick sort to sort the array in ascending order. At each step indicate the pivot element you have selected.

Steps must be improperly considered in breadth on the recursion tree, rather than in depth. Return as a result the two partitions of the original array and the two partitions of those found at the previous step.



6 4 7 0 8 3 6 9 15 19 12 11 10 13 14



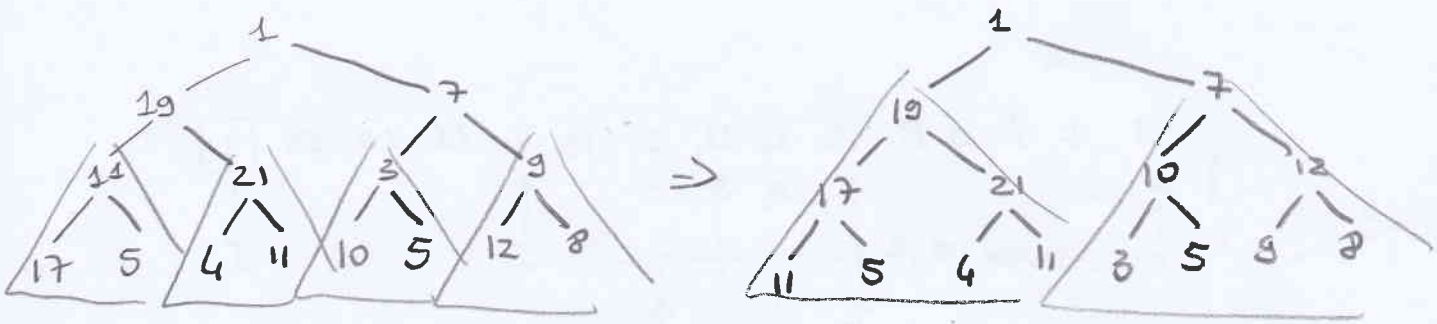


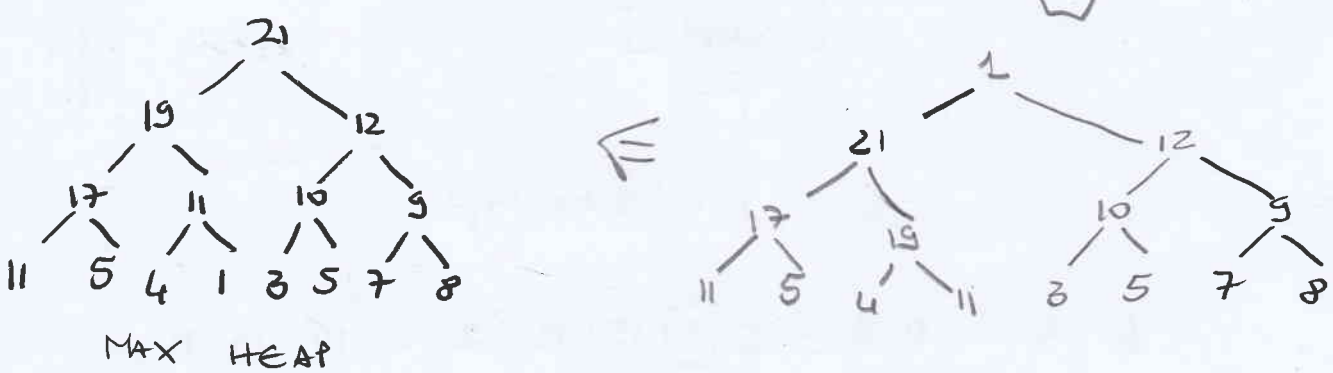
4. **(2.0 points)**
Given the following sequence of integers stored into an array:

1 19 7 11 21 3 9 17 5 4 11 10 5 12 8

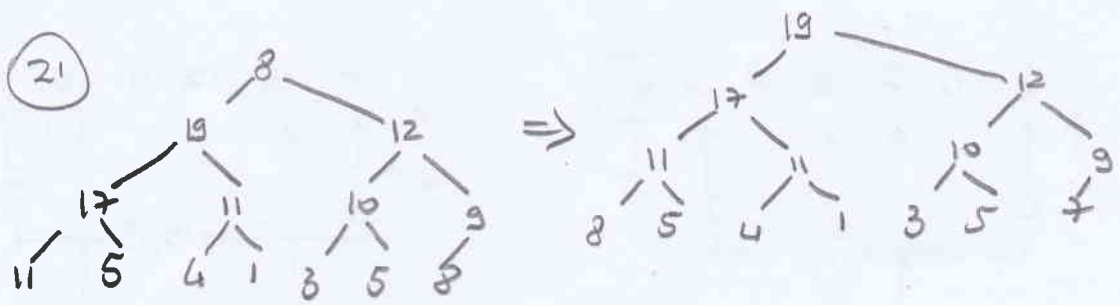
turn it into a heap, assuming to use an array as underlying data structure. Draw each step of the heap-building process, as well as the final result. Assume that, at the end, the largest value is stored at the heap's root. Execute the first three steps of the heap sort algorithm on the heap built at the previous step.

Show all relevant intermediate steps.

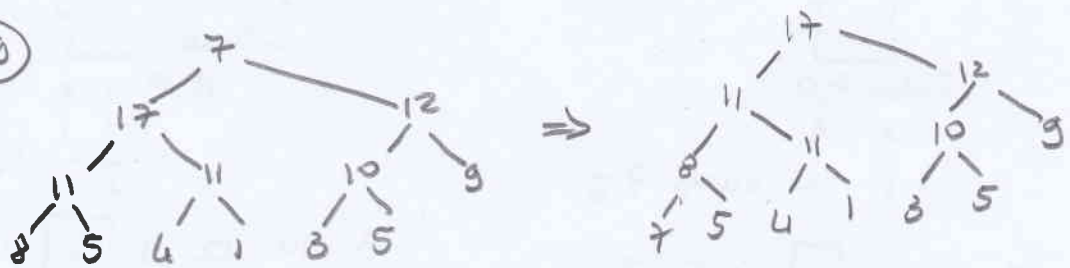




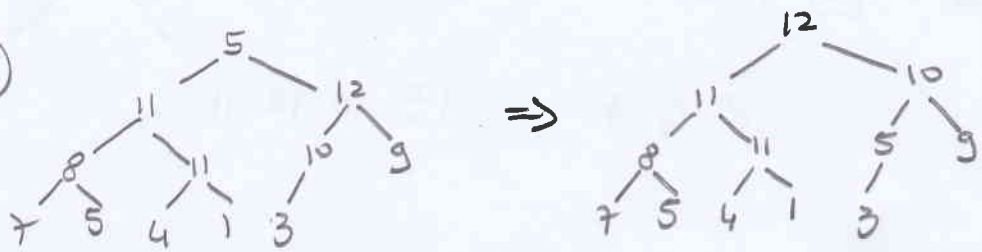
HEAP SORT
1° STEP



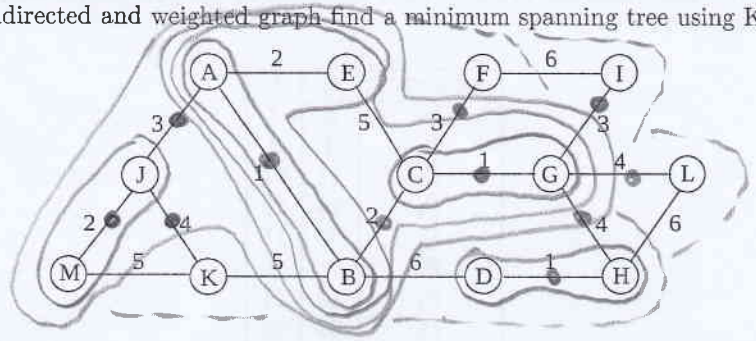
2° STEP



3° STEP



5. (2.0 points)
10 credit course (01*OGDLP*)
Given the following undirected and weighted graph find a minimum spanning tree using Kruskal's algorithm.



Draw the tree and return the minimum weight as a result. Show all relevant intermediate steps and all cuts generated.

Define and explain what a "cut", a "crossing edge", and a "set of edges respecting a cut" are. Report an example to illustrate these three concepts.

AB	1	✓
CG	1	✓
DH	1	✓
AE	2	✓
BC	2	✓
JM	2	✓
AJ	3	✓
CF	3	✓
GI	3	✓
GH	4	✓
GL	4	✓
JK	4	✓
EC	5	NO
BK	5	NO
MK	5	NO
BD	6	NO
FI	6	NO
HL	6	NO
MST WEIGHT	30	

Given $G=(V,E)$

CUT: A partition of V into S and $V-S$ such that
$V = S \cup (V-S)$ && $S \cap (V-S) = \emptyset$

CROSSING EDGE: An edge $(u,v) \in E$ which crosses a cut, i.e., such that $u \in S$ && $v \in (V-S)$ or viceversa

SET of EDGES RESPECTING a CUT:
A cut respects a set of edges A if no edge of A crosses the cut

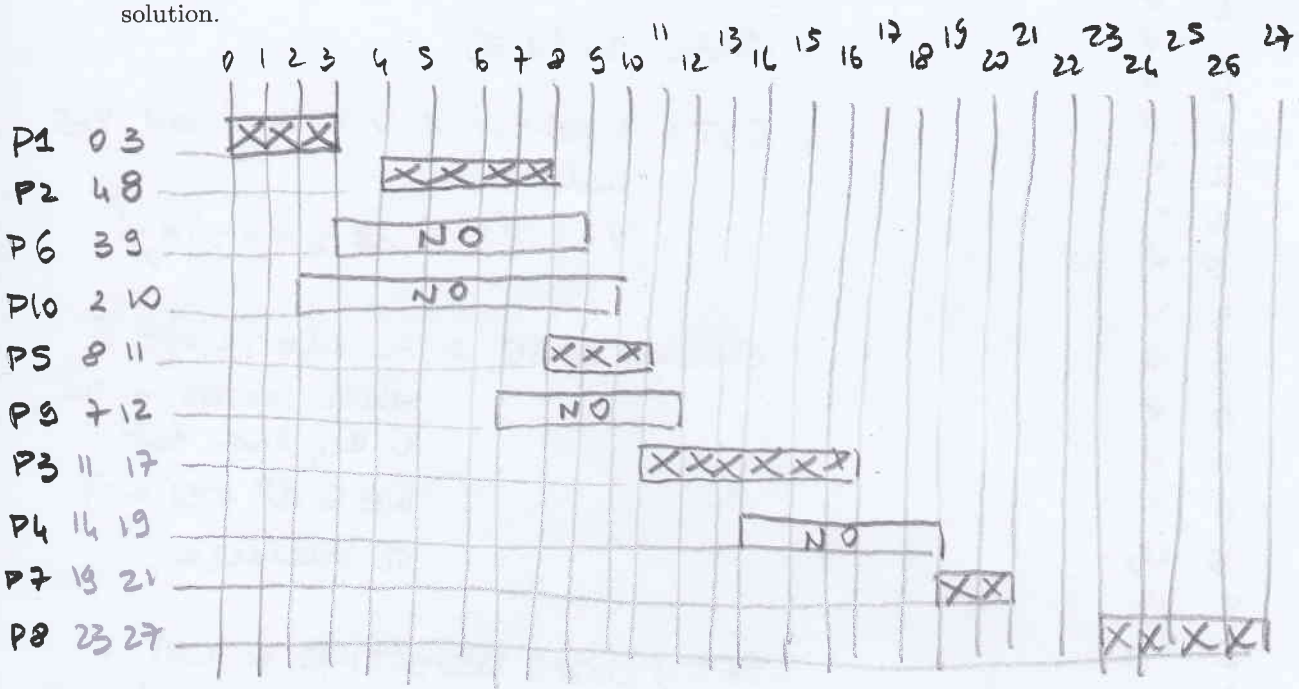
12 credit course (*02OGDLM*)

Given the following activity set, where the i-th activity is identified by the pair $[s_i, f_i)$ where s_i is the starting time and f_i is the finishing time:

ACTIVITY	s_i	f_i
P_1	0	3
P_2	4	8
P_3	11	17
P_4	14	19
P_5	8	11
P_6	3	9
P_7	19	21
P_8	23	27
P_9	7	12
P_{10}	2	10

Using a greedy algorithm, find the largest subset of mutually compatible activities.

Explain what "greedy algorithms" are, and provide a few examples of greedy algorithms achieving an optimal solution.

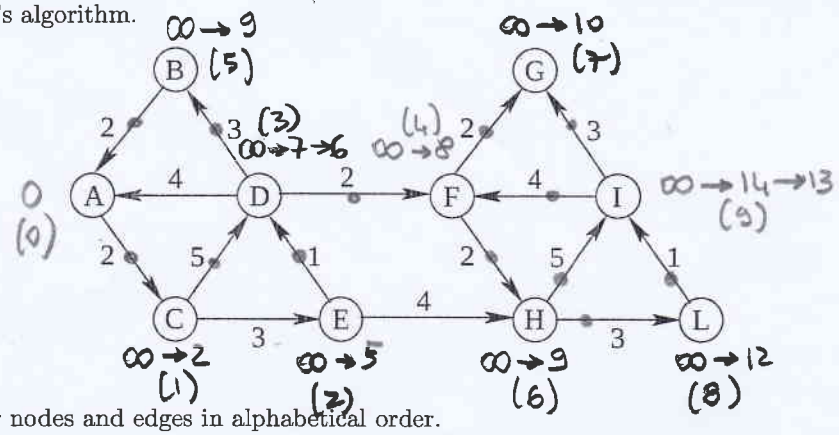


Compatible activities: P1, P2, P5, P3, P7, P8

GREEDY ALGORITHMS: make the choice that look best at the moment. They make locally optimal choices in the hope that these choices will lead to a globally optimal solution.
Do not always yield optimal solutions

EXAMPLES: activity selection, Huffman code, MST Kruskal & MST Prim.

6. (2.0 **points**)
Given the following directed and weighted graph, find all shortest paths connecting node A with all the other nodes resorting to Dijkstra's algorithm.



If necessary, consider nodes and edges in alphabetical order.

Would Bellman-Ford's algorithm find the same result? Justify your answer.

A	B	C	D	E	F	G	H	I	L
0	9	2	6	5	8	10	9	13	12

As there are no negative weight edges and no negative weight cycles both Dijkstra and Bellman-Ford find the shortest paths on this graph.