On Interval Trees

You need to be alert to (usually minor) changes that may be made to the assignment statement or to the guidelines after the assignment is first put up. Refresh this frame and re-read the assignment carefully before you make your final submission.

Interval trees

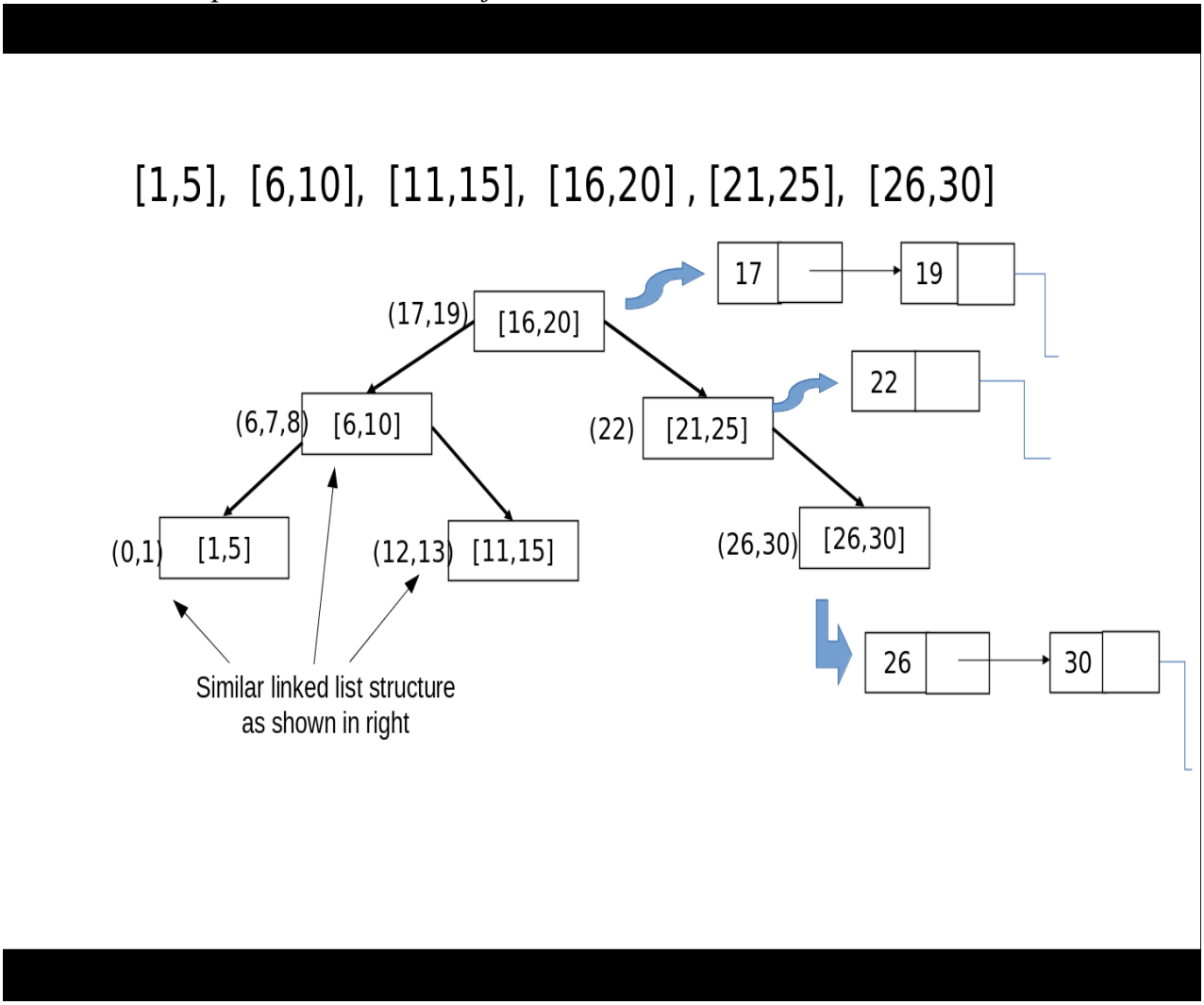
We consider an Interval Tree as a binary tree where,

1. Each node is labeled with an integer interval range $[a, b]$ where $a \leq b$.
2. For a node a labeled with $[l_a, u_a]$ in the tree, any node b occurring in the left subtree of a with label $[l_b, u_b]$ shall imply that $u_b \leq l_a$.
3. For a node a labeled with $[l_a, u_a]$ in the tree, any node b occurring in the right subtree of a with label $[l_b, u_b]$ shall imply that $u_a \leq l_b$.

Consider the example interval tree given here with the special property that

1. all intervals are mutually $exclusive$ and $exhaustive$ for the entire range [0,30].
2. a node a labeled with $[l_a, u_a]$ points to a linked list containing elements $i, j, \cdots, k$ in sorted order (where $l_a \leq i, j, \cdots, k \leq u_a$).

Let us call this special structure as *disjoint interval list*.

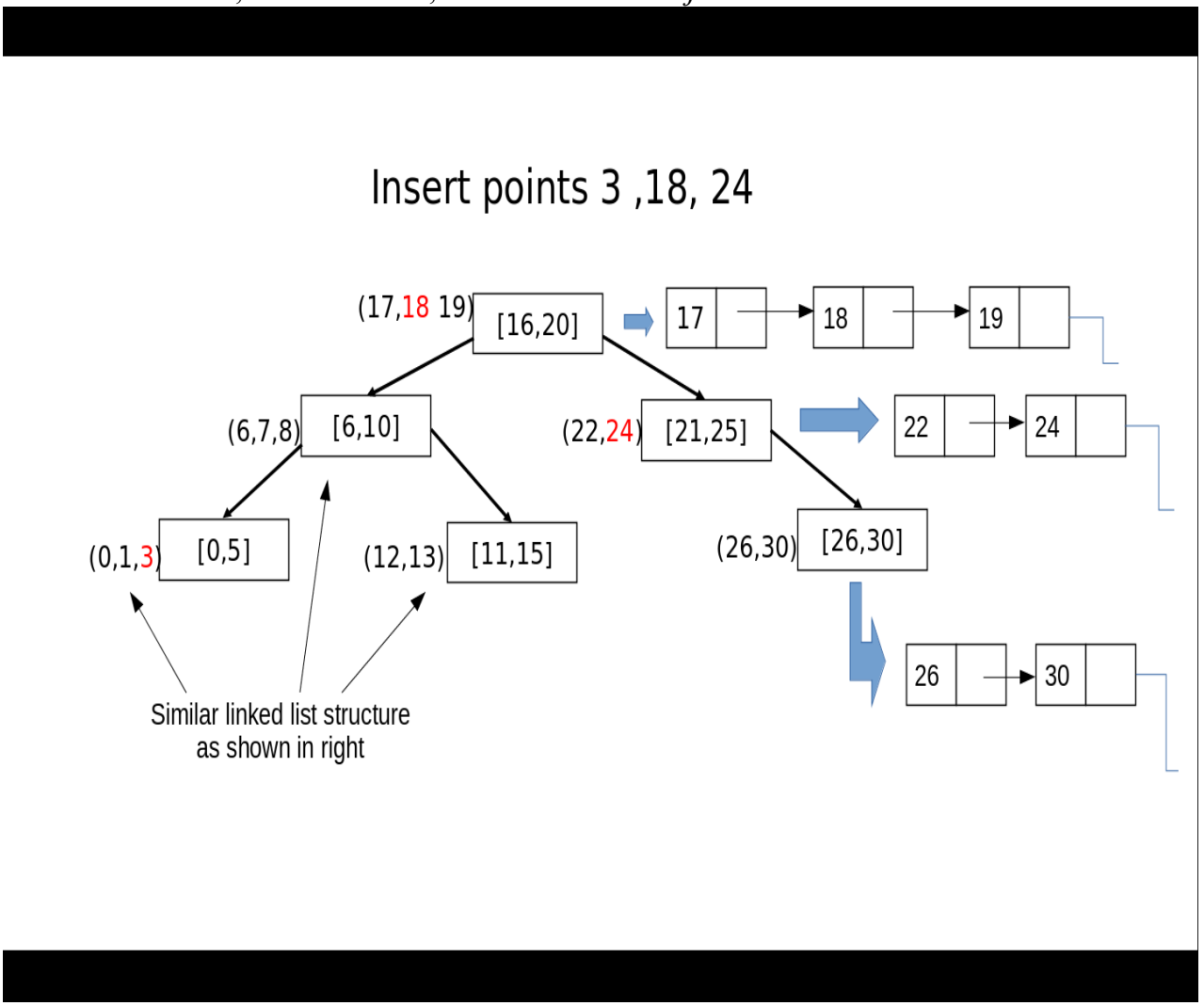


Assignment statements

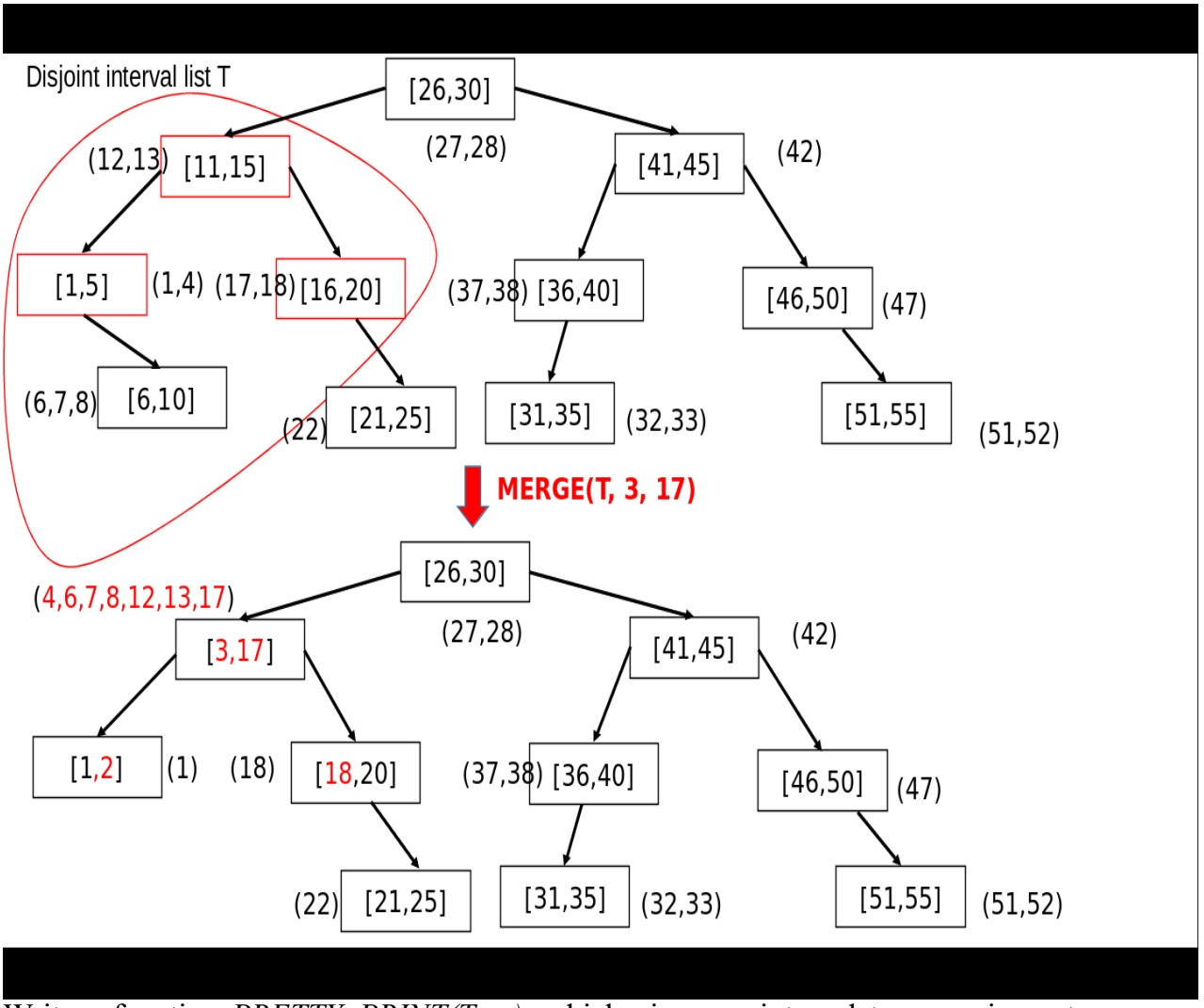
1. Write a function *CREATE_INTERVAL_TREE(l,u,n)* which creates *n* number of mutually exclusive and exhaustive equal sized intervals within the interval range [l,u].Surely, it is not always possible to maintain such equal sized intervals when we are dealing with integers. In such cases, let the boundary intervals be (a bit) smaller or larger w.r.t. other intervals that are of equal size. Generate a *disjoint interval list* given this upper and lower bounds and number of intervals.

   To make the interval tree hight balanced, first create a root by finding out the median of the lower bounds of the intervals and select the corresponding interval as root node. Recursively generate a left subtree from the intervals lesser than the root interval and a right subtree from list of intervals greater than the root interval.

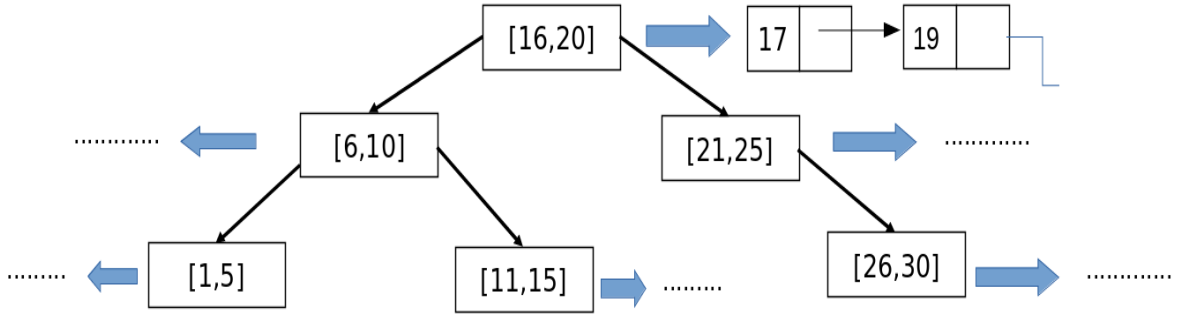
2. Write a function *INSERT(Tree, i)*, which inserts an integer i to the *disjoint interval list* $Tree$. This means, after insertion, we have a valid *disjoint interval list*.



3. Write a function *MERGE(T, l, u)* that takes as input a disjoint interval list T and two integers $l \leq u$. The function will do the following
   1. create a node N with label $[l, u]$.
   2. identify the *unique* subtree T' in T with intervals inside or overlapping with $[l, u]$.
   3. traverse T', create a linked list pointed to by N and populate it with elements from T'.
   4. modify pointers to delink T' from T and add N



4. Write a function *PRETTY_PRINT(Tree)*, which given an interval tree, carries out an inorder traversal of the tree and outputs the data values contained in the linked lists of each node in the format where the nodal lists are printed at every line and the node at level *h* has *h* horizontal tabs before it.




```
[1,5] ........
      [6,10] .........
[11,15] ..............
              [16,20] 17, 19
       [21,25] ..................
[26,30] ................
```

The higher the level of a node, the greater the space during printing.

5. Write a main() which does the following using the helper functions as discussed above.
    1. Ask user to provide l, u, n and call *CREATE_INTERVAL_TREE(l,u,n)*.
    2. Ask user to provide integer i, generate i random integers inside $[l, u]$ and insert them to the created *disjoint interval list*.
    3. Ask user to provide integer $m < n$. Write a suitable routine which modifies the existing *disjoint interval list* to another *disjoint interval list* which should have m number of mutually exclusive and exhaustive equal sized intervals within the interval range [l,u]. This job should be entirely performed by suitable calls to $MERGE$.
    4. Print the data structure using PRETTY_PRINT after each step

Submissions

1. C-program file.

   NB: Assignment evaluation will be automatic, so do ensure that the output formatting specifications are strictly followed.

   Do not output anything apart from what you have been asked to output, as that will confuse the automatic evaluation mechanism and as a result you will end up getting lower marks.

2. A report outlining pseudo code and complexity analysis of *MERGE* and *CREATE_INTERVAL_TREE* routines.

   A sample tex file is also made available. That should be compiled with the command `pdflatex rep.tex` or just `pdflatex rep`

You can also work on this file using visual latex packages available on both linux and windows.

Marking Guidelines

Assignment marking is to be done only **after** the deadline expires, as submissions gets blocked after the assignment is marked.

CREATE_INTERVAL_TREE()	**10**
INSERT()	**5**
MERGE()	**10**
PRETTY_PRINT()	**5**
main()	**10**
Report	**10**
Total Marks	***50***

Assignment submission

You need to submit a program file for the assignment and the report and also enhanced/optimised versions of routines (individual submission).

Use electronic submission via the WBCM link

You should keep submitting your incomplete assignment from time to time after making some progress, as you can submit any number of times before the deadline expires.

Warning

Cases of copying will be dealt with seriously and severely, with recommendation to the Dean to de-register the student from the course.