* Anytune you use heap (dynamic memory) it is the law to use the big three
  + Copy constructor
  + Assignment operator (overloaded)
  + Destructor
  + Otherwise you'd have memory leaks.
* Set data structure
  + #include <set>
  + Set<int> (template class) s,t,u;
  + s.insert(30);
  + s.insert(10);
  + s.insert(40);
  + s.insert(20); //s = {30,10,40, 20}
  + t.insert(1);
  + t.insert(2);
  + t.insert(3); //t={1,2,3}
  + Set\_union(s.begin(),s.end(), t.begin(), t.end(), u.begin())
  + For (set<int>::iterator I = u.begin(); I != u.end(); I++)
    - Cout << \*I << endl;
* Binary Tree – Tree
  + A tree is an acyclic graph
  + A graph consists of a set of nodes and a set of edges.
  + An edge connects a pair of nodes.
  + Ex: G1 A------B ------- C therefore there is no edge connecting A and C directly
  + In a Tree one of the nodes is designated as the root. Fig 13.1 and 13.5
  + In a binary tree no node has more than two children. Fig 13.2, 13.4, 13.6, 13.7, 13.8, 13.9
  + In a complete binary tree all nodes are pushed up and left.
    - A complete binary tree of height h has between 2^h and 2^(h+1)-1
    - Or a complete binary tree containing n nodes has a height = |\_log2n\_|

|  |  |
| --- | --- |
| n | 2^n |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |
| 5 | 32 |
| 6 | 64 |
| 7 | 128 |
| 8 | 256 |
| 9 | 512 |
| 10 | 1024 |
| 20 | 2048 |
| 30 | 4096 |

* Floor function
  + └2.1┘ = 2
  + └2.00000095┘ = 2
  + └2.99999┘ = 2
  + Drops the faction and leaves the largest integer.
* Ceiling Function
  + ┌2.1┐ = 3
  + ┌2.09┐ = 3
  + ┌2.999┐ = 3
* In a height balanced binary tree at every node the difference between heights of right and left subtrees is within one.
* A null pointer(no subtree) has height –1.
* Height of a height balanced binary tree with m(n) nodes is in o(logm(n)).
* Set will be a height balanced binary search tree.
* In order traversal
  + LNR – left node right
  + 5 – 10 – 15 – 20 – 30 – 35 – 40. Always prints out a tree in order.