Problem P.1

Ans 4) Vector, List, Deque, Set, Multiset, Map, Multimap all use the insert method.

Problem P.2

We can define class members static using “static” keyword. When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member. A static member is shared by all objects of the class. All static data is initialized to zero when the first object is created if no other initialization is present. We cannot put it in the class definition, but it can be initialized outside the class as done.

https://www.tutorialspoint.com/cplusplus/cpp_static_members.htm

static int objectCount;

int Box::objectCount = 0; // Initialize static member of class Box outside the class

static members are special members whose memory is automatically allocated at compile time and it's scope is entire program but only visible in function which is defined.

static members doesn't require an object to access it. it can call directly using class name.

now answer for your question , objects require separate memory.

for example a class of student, aruns object and vishak object is different, because they have different characteristics. so they have separate memory for each object.

But in the case of static members the only one memory location is available. and it is allocated in compile time. that's why external declaration is given to that.

static members are actually not a part of instance of class, object. It is a part of class or program that's why it need external declaration.

In the question we have:

```
template <class T>
class Test
        private:
                T val;
        public:
                static int count;
                Test()
                        count++; //every time a Time object is created the value of count increases by 1

template<class T>
int Test<T>::count = 0; // here the static member is initialized outside the class with value 0

int main()
        Test<int> a;// creating an Int object a
        Test<int> b;//creating another Int object b
        Test<double> c;//creating a double object c
        cout << Test<int>::count<< endl;//since two int object has been declared;value of count is 2
        cout << Test<double>::count << endl;//only 1 double has been declared;value of count   is 1
```

Answer

Output:

2

1

Problem P.3

https://www.geeksforgeeks.org/passing-by-pointer-vs-passing-by-reference-in-c/

```
template<class T>
Void multiples (T &sum, T x, int n)
{
        Sum=0;
        for(int i=1;i<=n;i++)
                sum=sum+ (i*x);
}
```

Problem P.4

I think all because for searching the time complexity of following is:

Heap- O(n) must go through all elements

Max-Heap- O(n) must go through all elements because the property does not provide much help

Queue- O(n)

BST- O(h) Best case O(logn) Worst case O(n)

RBT- O(logn)

0(nlogn) satisfies the upper bound for all the above data structures.

Problem P.5

All false except because after using master theorem we get $\theta(n^{log2(3)})$ which does not match any options.

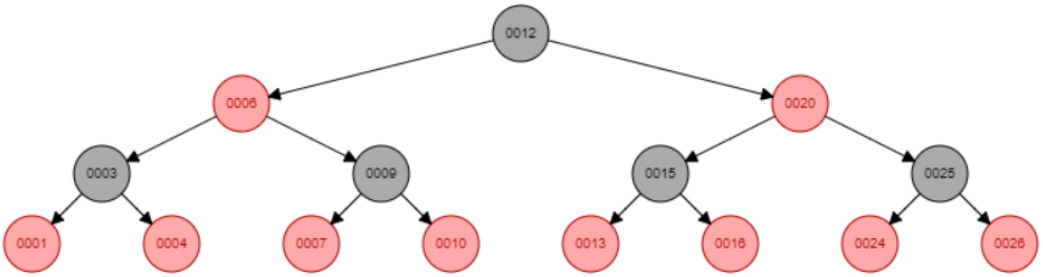
Problem P.8 Randomized Quicksort

```
Randomized_Partition(int* A, int l, int r)
        int i=RANDOM (l,r);
        swap A[i] with A[l]
        return Partition(A,l,r)
Partition(int* A, int l, int r)
        X=A[l] //pivot
        K=l;
        for(int i=l+1 to r)
                if(A[i]<=X)
                        k++
                        swap A[k] with A[i]
        swap A[k] with A[l]
        return k
Randomized_Quicksort(int* A, int l,int r)
        if(l<r)
                q=Randomized_Partition(A,l,r)
                Randomized_Quicksort(A,l,q-1)
                Randomized_Quicksort(A,q+1,r)
```

Problem P.9

RBT with height 3

The height of a binary tree is the largest number of edges in a path from the root node to a leaf node. Essentially, it is the height of the root node. Note that if a tree has only one node, then that node is at the same time the root node and the only leaf node, so the height of the tree is 0. On the other hand, if the tree has no nodes, it's height is -1.



Problem P.6

Karatsuba algorithm

Let N1 and N2 be two numbers.

Then we can write them as:

N1= $(a*10^{n/2}) + b$ where n=total digits in the number a is the first part and b is the second part of the number

N2=$(c*10^{n/2}) + d$

Then,

N1*N2 = $(a*c)*10^{n} + (a*d)* 10^{n/2} + (b*c)*10^{n/2} +(b*d)$

= $(a*c) *10^{n} +(ad+bc) *10^{n/2}+(b*d)$

=$(ac)* 10^{n} + ((a+b)*(c+d)-ac-bd) *10^{n/2} +(bd)$

There we need to do three recursion for the three multiplications ac, bd and (a+c)*(c+d)

4 additions and 2 subtractions

2 bit-shifting done from 10^{n} and $10^{n/2}$

T(n)= 3*T(n/2) + n (assuming all other operations can be done in linear time)

By using master theorem, we get the following result T(n)= $\theta(n^{\log2(3)}$

```
Function Karatsuba (num1, num2)
 if (num1 < 10) or (num2 < 10) // base case of recursion when each number is a single digit
      return num1 × num2
   //Calculates the size of the numbers
   m1= log10(num1) + 1  //size of first number
   m2= log10(num2) + 1 //size of second number
   n= min (m1, m2)
   m= ceil(n/2)

   //Splitting the digit sequences in the middle
   a= num1 div 10^m
   b= num1 mod 10^m
   c= num2 div 10^m
   d= num2 mod 10^m

   //recursive steps
   s1 = Karatsuba (b, d) // b*d
   s2 = Karatsuba ((a + b), (c + d))// (a+b) *(c+d)
   s3 = Karatsuba (a, c) // a*c

   return (s1 × 10 ^ (n)) + ((s2 - s3 - s1) × 10 ^ m) + s1
```

Problem P.11 Breadth First Search

All the distance except the source node is set to infinity and the color is set to WHITE

The color of the Source Node 0 is set to Grey and its distance is set to 0.

Then the source node is added to the queue using the enqueue function.

Now we run the main iterative part and the procedure runs as the following:

Step 1: We take out Node 0 from the Queue using dequeue function.

Step 2: We visit all the connected nodes of Node 0 which have not been visited before (i.e their color is white) and these are Node 1 and Node 2. Let us suppose we visit Node 1 at the beginning thein its color is set to GREY and the distance of it is updated by the formula V.d= U.d +1 where V is the current node and U is the parent node. Then Node 1 is added to the Queue. Then we visit Node 2 and do the same steps as above for this and add it into the queue as well. Finally the color of the Node 0 is set to be BLACK.

Step 3: Now we remove another element from the queue which is Node 1 as it was added before Node 2. We check if it has any connected nodes/edges. It has two connected nodes and they ae Node 2 and Node 3. Since we had already visited Node 2 and its color is set as grey currently so now, we visit Node 3. We set the color of Node 3 to grey and update its distance to 3 as U.d is 2 here. We add Node 3 into the queue and finally set the color of Node 2 to BLACK.

Step4: Now we remove Node 2 from the queue and check for its neighboring Nodes. We see that it has only one connected node which is 3 and we have already visited it so we don't do anything for Node 2 just change its color to BLACK.

Step5: Now we remove Node3 from the queue and check for its neighboring Nodes. It has one neighboring Node 4 so we visit that since its color is still white. We set the color of Node 4 to 0 and update its distance to 4 since U.d was 3 here. Now we add Node 4 to the queue and set the color of Node 3 to black.

Step6: Now we remove Node 4 from the queue and check for its neighboring unvisited nodes. It has only one unvisited node that is Node 5, so we visit that and set the color to grey. We update the distance of Node 5 to 5 since U.d was 4. Now we add node 5 to the queue and change the color of Node 4 to BLACK.

Step 7: We remove Node5 from the queue and check for its neighboring nodes. Since it has no neighboring nodes, we set its color to BLACK. Now the queue is empty to the main loop ends and the BFS algorithm is finished.

Problem P.7

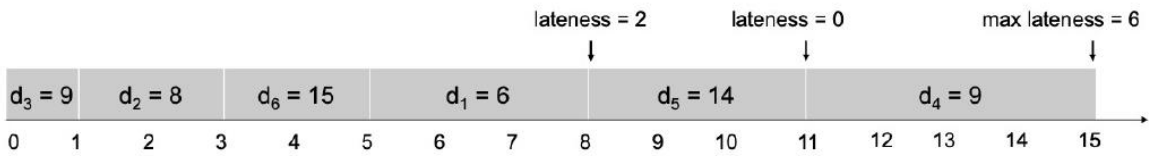
```
Void transform (node** root)
{
        Node*temp;
        Node*max;
        int i=0;
        while(*root!=NULL)
        {
                if(i==0)
                        max=temp=extract_max(root);
                else
                        if(i%2==0)
                                replace(root,temp->right,extract_max(root))
                                temp=temp->right
                        else
                                replace(root,temp->left, extract_min(root))
                                temp=temp->left
                i++
        }
        *root=max;
}
```

Problem.10

a)

Ans

Greedy choice: Shortest processing time first – Fails because of the following example:



Here each job is selected according to smallest processing time, but we can see that the lateness is 2 for Job 1 and 6 for Job 4. So, this greedy choice fails.

Next greedy choice: Smallest slack d_j-t_j is chosen first. Fails because of the following example

	1	2	3	4	5	6
t_j	3	2	2	4	3	1
d_j	6	8	9	9	14	15

i	d_i-t_i
1	3
2	6
3	7
4	5
5	11
6	14

So, the jobs are chosen in the order 1 4 2 3 5 6

Maximum lateness here is 2 for Job 3 so not globally optimal solution.

b)

Greedy choice for optimal solution is to select the order of Jobs according the earliest deadline/ due time.

Pseudocode:

Let n be the total number of jobs then,

Sort Jobs (processing time t[j]) according to their due time such that d[1]<= d[2]<=d[3]<=…d[n]

S.time=0

for (j= 1 to n)

    Output[j].start_time= S.time

    Output[j].finish_time= S.time + t[j]

    S.time= S.time+ t[j]

return Output

d[j] – Sum of Output[j-1].finish_time and Output[j].finish_time <=1 so it is correct.