depth of node x-number of edges on path from root to x

height of node x-number of edges on longest path from leaf to x

height of tree-height of root node

binary tree-each node has at most two children

applications of binary tree

storing naturally hierarchical data

storing data for quick search, insertion and deletion (binary search)

trie – dictionary

networking

|  |  |  |  |
| --- | --- | --- | --- |
| Structure | Operation | Time | Space |
| Array | Linear Search | O(n) |  |
|  | Insertion | O(1)in not full, O(n) if full |  |
|  | Deletion | O(n) |  |
| Sorted Array | Binary Search | O(log n) |  |
|  | Insertion | O(n) to shift numbers for insertion |  |
|  | Deletion | O(n) to shift numbers after deletion |  |
| Linked List | Search | O(n) |  |
|  | Insertion | O(1) at head  O(n) at tail |  |
|  | Deletion | O(n) |  |
| Balanced Binary Search Tree | Search | O(log n) average case |  |
|  | Insertion | O(log n) average case |  |
|  | Deletion | O(log n) average case |  |
| Unbalanced Binary Search Tree | Search | O(n), worst case |  |
|  | Insertion | O(n), worst case |  |
|  | Deletion | O(n), worst case |  |

void insertion\_sort (int myArray[], int length){

int hole, value;

for (int i = 1; i <=length-1; i++){

value=myArray[i];

hole=i;

while (hole>0 && myArray[hole-1]>value){

myArray[hole]=myArray[hole-1];

hole=hole-1;

}

myArray[hole]=value;

}

}

int binary\_search (int A[], int n, int x){

int low=0;

int high=n-1;

while low<=high

{

int mid=(low+high)/2;

if (x==A[mid]) return mid;

else if (x<A[mid])

high=mid-1;

else

low=mid+1;

}

return -1;