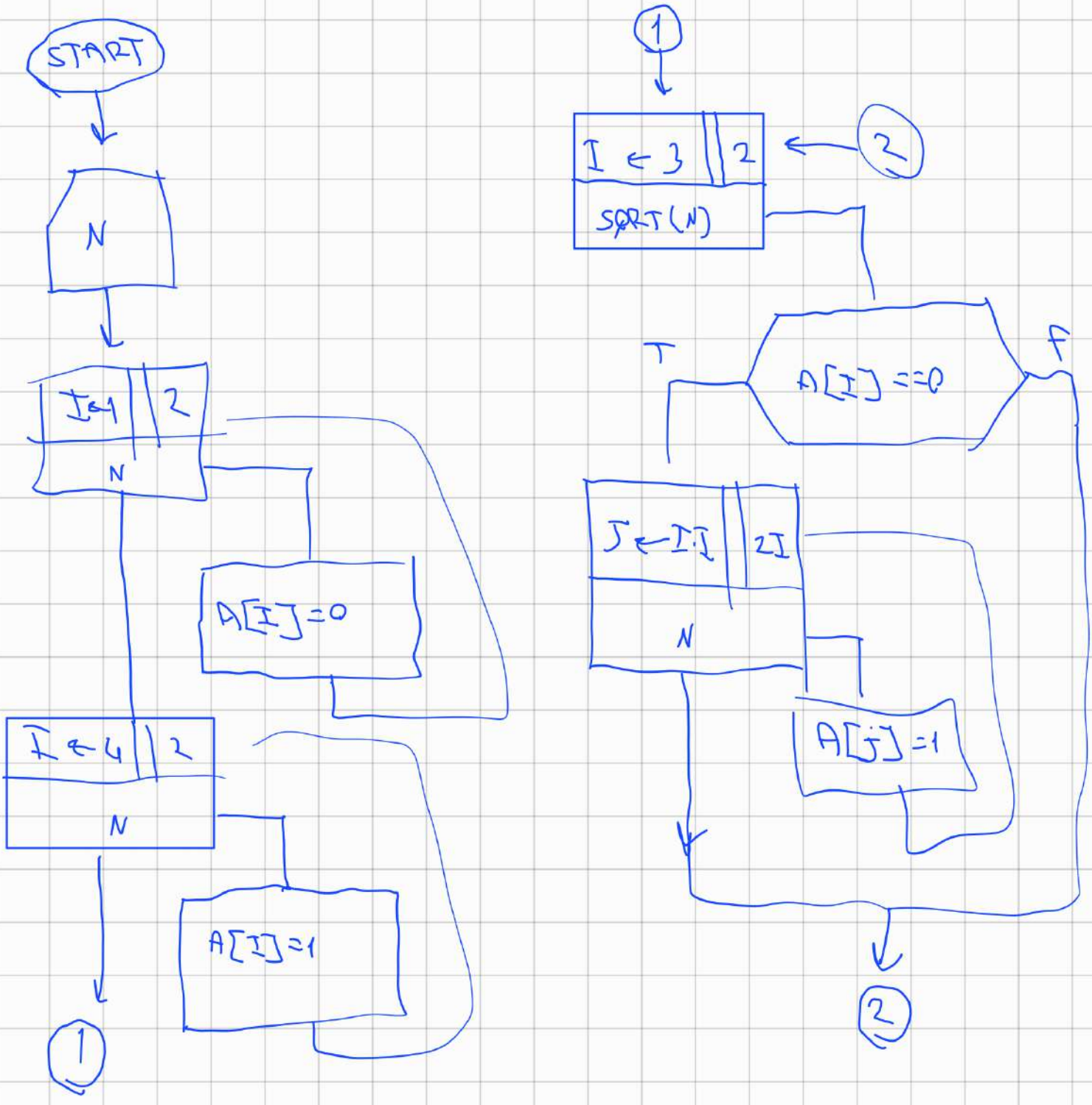
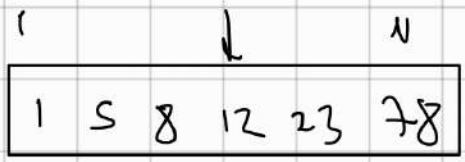
n sayısının asal olup olmaması



Search Algorithms

- Linear search
- Binary search
- Binary search tree
- Interpolation search
- Exponential search

- Hashing
- Jump Search



- Sorted
- unsorted

Decrease and Conquer

Binary Search $O(\log_2(N))$

1	5	2	4	3

Linear Search $O(N)$

Algorithm Analysis

- Time complexity
- Space complexity

→ worst case
→ average case
→ best case

Basic operation

- Comparison operation
- Assignment operations (Atama)
- Arithmetic operations
- Initialization

Example Pseudo code (Finding Maximum Element)

	Basic operation
currentMax ← A[0]	2
for i ← 1 to n-1 do	2+n
if A[i] > currentMax then	2.(n-1)
currentMax ← A[i]	2.(n-1)
	2.(n-1) → increment counter
return currentMax	1

Total $7n-1$ $O(7n-1)$

$\rightarrow O(n)$

Jump Search

1 5 8 12 28 32 42 53 68 72 73

42 aranıyosa $\sqrt{n}$ birim ileri gidilir. Gidilen değer 42den büyükse ilk elemana geri dönüp $\sqrt{n}$. elemana kadar Linear search yapar. küçükse $\sqrt{n}$. elemandan $\sqrt{n}$ birim elemana kadar search yapar.

Interpolation Search Nedir

2, 4, 6, 8, 10, 12, ----, 100

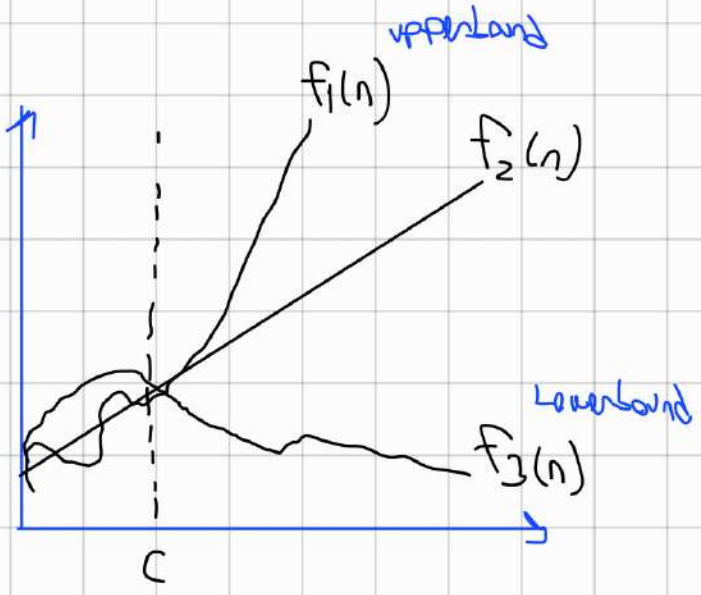
Eğer bir dizi linear olarak artış göstriyorsa bu linear formule göre search yapılır.

Asymptotic Notation

Big oh (O) notation : upperbound

Big Theta (Θ) not. : Tight bound

Big Omega (Ω) not. : Lowerbound



Functions Growth rate

$1 \rightarrow \log(N)$ N $N\log(N)$ N^2 N^3 2^N $N!$

Linear Search Algorithm

1, 3, 5, 2, 4 — O(1) → best case
O(n) → worst case

1, 2, 3, 4, 5 — O(1) → best case
O(n) → worst case

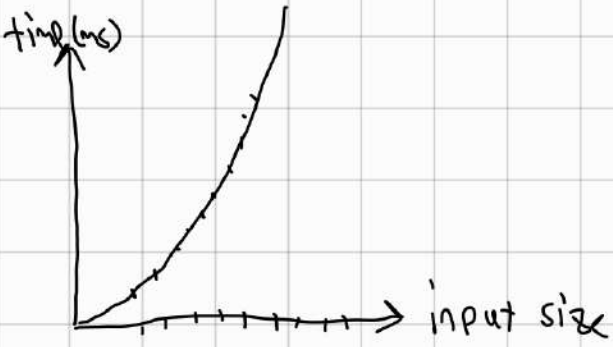
Max-Min element array

unsorted: O(N) → best case
O(N) → worst case

Sorted: O(1) → best case
O(1) → worst case

Time complexity: Number of Inputs (N), Type of Inputs

- Theoretical Analysis
- Experimental Evaluation



Recursion vs Iteration

Criteria	Iteration	Recursion
- mode of implementation	using Loops	Function call itself
- State	Defined by the control variable's value	Defined by the parameter values stored in stack
- Progression	The value of control variable moves toward	the function state converges toward the base case

	the value in condition	
- Termination	Loop ends when control variable's value satisfies the condition	Recursion ends when base case becomes true
- Code Size	Iterative code tends to be bigger in size	Recursion decreases the size of code
- No termination state	Infinite Loops use CPU cycles	Infinite Recursion may cause stack overflow error or it might crash the system
- Execution	Faster	Slower

Factorial Program using Recursion

```
int factorial (int n){

  if (n<=1) // base case
  return 1;
  return n*Factorial(n-1);
}
```

x^y Recursion

```
int usingAlmn (int x, int y){

  if (y>=1)
```

```
    return x * yStHanoi(x, y-1);
  return 1;
}
```

Basamak bulma Recursion

```
int basamakBul(int n) {
  if ((n/10) == 0)
    return 1;
  return 1 + basamakBul(n/10);
}
```

İterasyon

```
- - - - - - {
static int count = 0; // fonk. birden fazla çağrıldığında değeri değişmez.
  if (n > 0) {
    count ++;
    basamakBul(n/10);
  }
return count;
```

Fibonacci Numbers

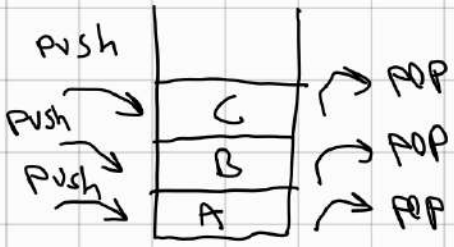
```
int fibonacci(int n) {
  int a, b;
    if (n <= 1)
      return n;
    a = 0; b = 1;
    for(i=2; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
}
```

Recursion version

```
int fib(int n){
  if (n <= 1)
    return n;
  return fib(n-1) + fib(n-2);
```

Stack (Abstract Data Type)

First in Last out or Last in first out



Applications: Evaluation of Arithmetic op.,

A+(B*c-(D/E*F) gi);

Recursive func. usage

function call

```
int main(){
f1()
}
```

```
void f1(){
f2()
}
```

```
void f2(){
f(3)
}
```

Reversing string, Backtracking (DeepFirst), Parenthesis matching

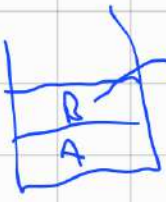
Conversion of decimal to other numb system

Stack operations

- Push

- POP

- Peek, top



get the value

- isFull - isEmpty

implemention

Stack → Array
↘ Linked List

Stack used in compilers

) a+b (error

a* (b+c)) error

Stack operations

```
#define MAX 10

typedef struct {
    int item[MAX];
    int top;
} STACK;
```

```
void initStack(STACK *s){
    s->top=0;
}
```

```
int isEmpty(STACK *s){
    if(s->top==0)
        return 1;
    else
        return 0;
```

```
int isFull(stack *s){
    if(s->top==MAX)
        return 1;
    else
        return 0;
}
```

```
int push(int x, STACK *s){
    if(isFull(s))
        return 0;
    else{
        s->item[s->top++]=x;

        return 1;
    }
}
```

```
int pop(Stack *s, int *x){
```

```
int peek(STACK *s, int *x){
    int adr;
```

```
if(isEmpty(s))
  return 0;
else{
*x=s->item[--s->top];
 return 1;
}
}
```

```
if(isEmpty(s))
return 0;
else{
adr= s->top -1;
*x= s->item[adr];
return 1;
}
}
```

Arithmetic Expression Evaluation

(3+5)*2/3+7 → infix

highest ↑

precedence	infix	prefix	postfix
(	A+B	+AB	AB+
*/	A+B*C	+A*BC	ABC*+
+-	A+B*C+D	++A*BCD	ABC*+D+

ex)

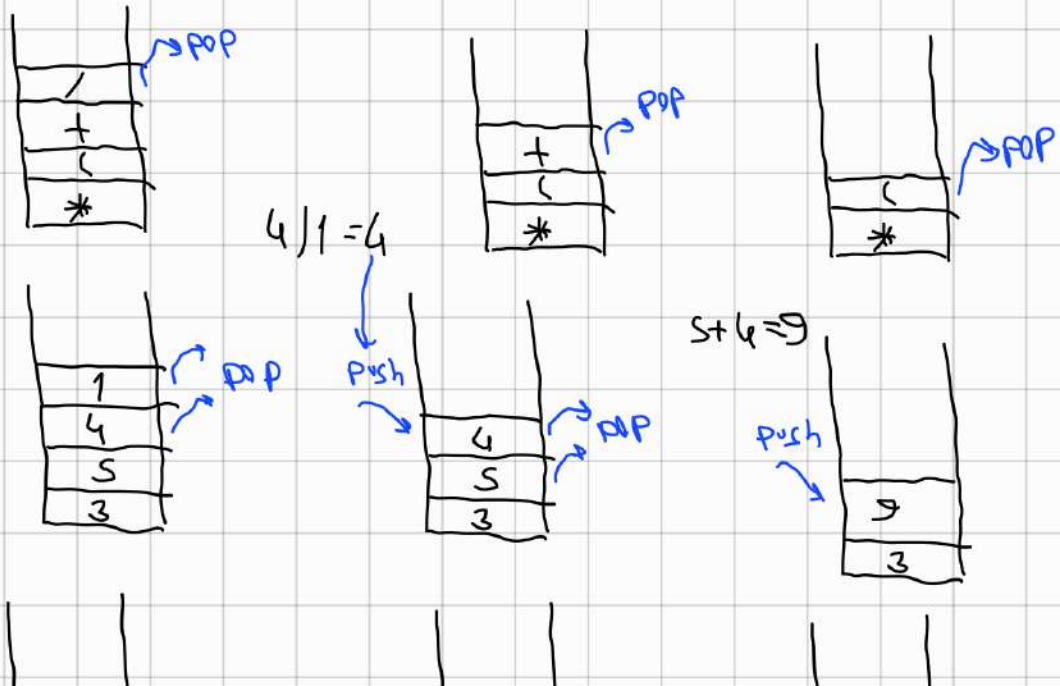
a+b*c-d/e*f → infix

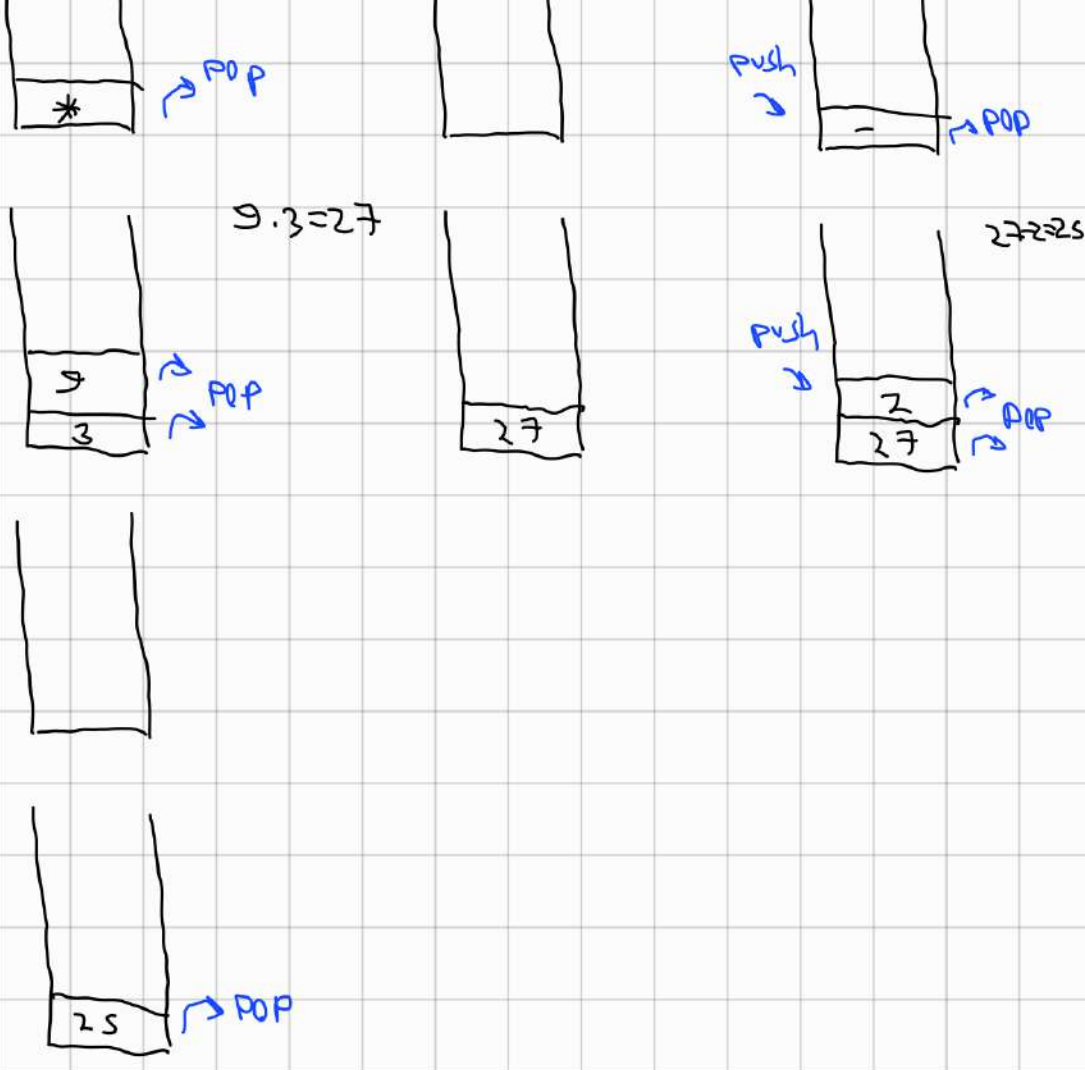
abc*+de/f*- → postfix

infix coming * postfix

3*(5+4/1)-2

3 5 4 1 / + * 2 -





Operator	Precedence	Associativity
Exponantion ^	Highest	Right Associative
* /	second highest	Left "
+ -	Lowest	Left "

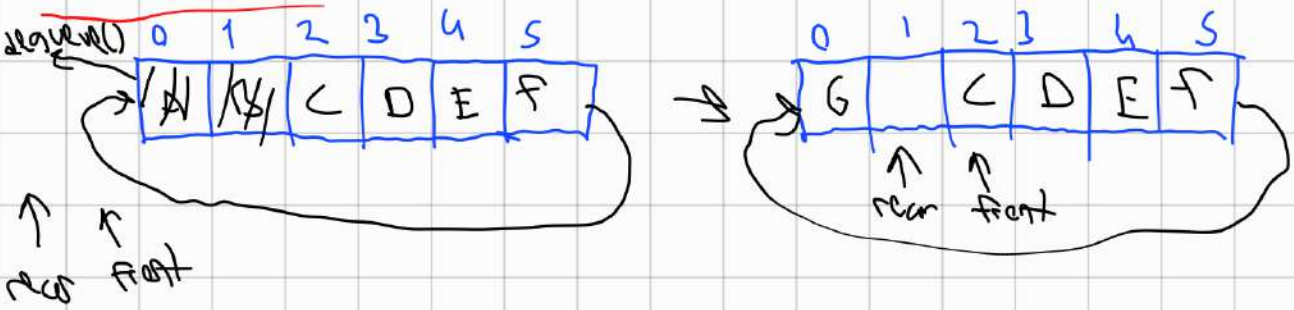
Queue (kuyruk)

Abstract Data type (soyut veri tipi)

first in first out

- circular Queue, Double ended Queue, priority Queue

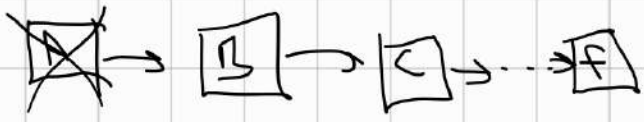
Circular Queue



Queue Operations

- isEmpty()
- isFull()
- enQueue()
- deQueue()
- initQueue()
- Peek()

implementation on Linked List



deQueue()
Free()

```
int isEmpty(){
  if (front == -1)
    return 1;
  else
    return 0;
}
```

```
int isFull(){
  if (rear == MAX-1)
    return 1;
  else
    return 0;
}
```

```
int deQueue(int *x){
  if (isEmpty)
    return 0;
  else{
    *x = q[front++];
  }
  return 1;
}
```

```
int enQueue(int value){
  if(isFull()){
    return 0;
  }
  else{
    q[++rear] = value;
  }
  return 1;
}
```

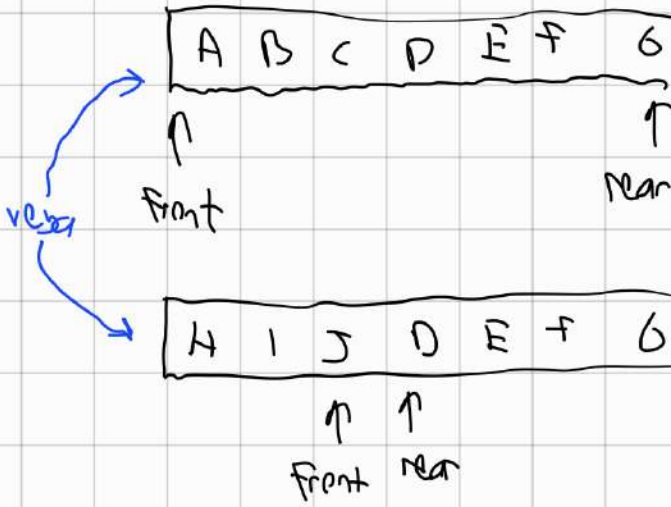
Circular Queue optional

```
void initQueue(){
  int front=-1, rear=-1;
}
```

```
int isEmpty(){
  if(front==-1)
    return 1;
  return 0;
}
```

```
int isFull(){
  if(front==rear+1 ||
  (front==0 && rear==SIZE-1))
  return 1;
  else
  return 0;
}
```

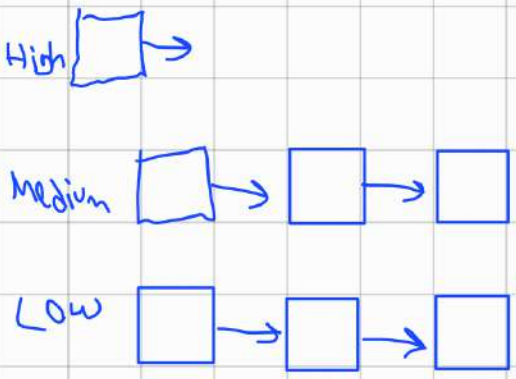
```
int enQueue(int value){
  if(isFull())
   return 0;
  else if(front==-1)
     front=0;
  rear=(rear+1)%SIZE
  q[rear]=value;
  return 1;
}
```




```
int deQueue(int *x){
   if (isEmpty)
     return 0;
   else{
   *x=q[front];
    if (front==rear){
      front=-1;
      rear=-1;
    }
    else
    front=(front+1)%SIZE;
```

```
        return 1;
    }
}
```

Priority Queue



Double Ended Queue

- Input Restricted DeQueue (Double ended Queue)
  input is restricted at a single end but allows deletion at both the ends
- Output Restricted DeQueue
  output is restricted at a single end but allows insertion at both the ends

Comparasion Stack versus Queue

	Stack	Queue
working Principle	LIFO	FIFO
Structure	same end used to insert and delete	one end is used for insert another end is used for deletion
Number of Pointers	1	2

operations	push, pop	Enqueue, Dequeue
variants	it doesn't have	circular, priority, Double ended Queue
Implementation	Simpler	Comparatively complex

Linear Data structure

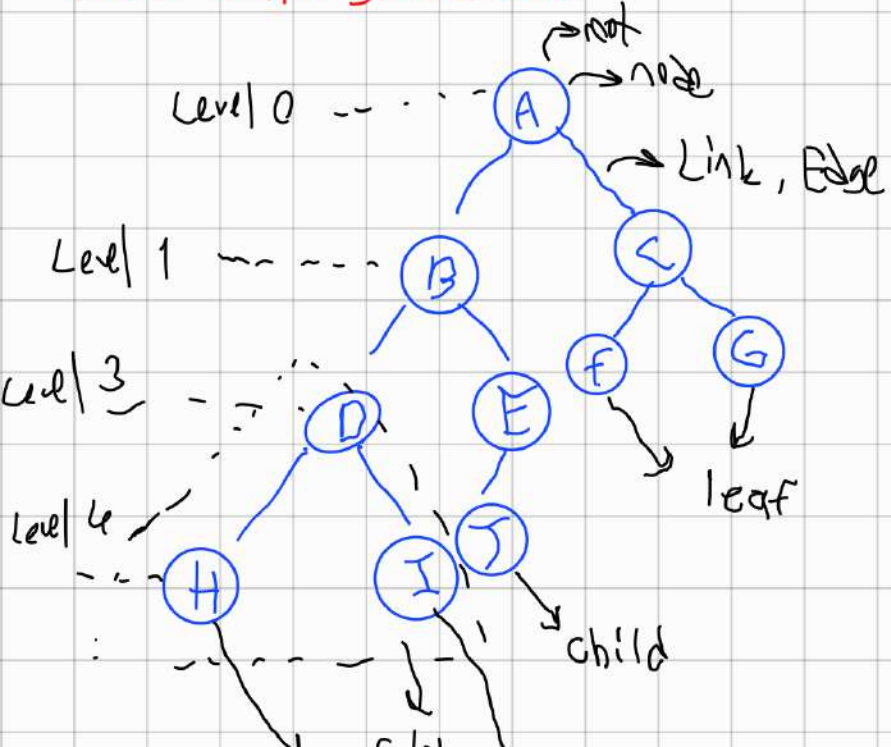
- Array
- Linked List
- Stack
- Queue

NON-Linear (hierarchical Data Structure)

- Graphs
- Tree

implementation	simpler	Complex
Levels involved	Single level	Multiple Level

TREE DATA STRUCTURE



Terminology

- Path
- Root
- parent
- child
- Leaf
- subtree
- Visiting
- Level
- forest

Subtree

Sibling node

- Depth
- Height

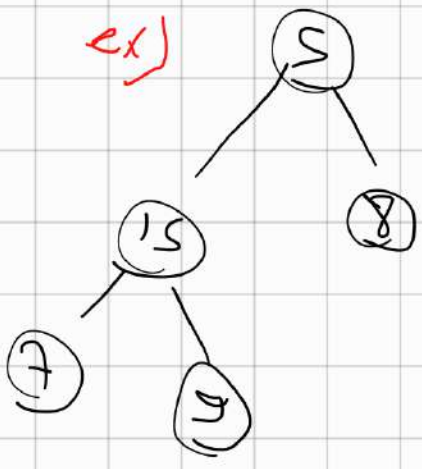
TREE APPLICATIONS

- File Management Systems
- Compression Algorithms (Huffman Algorithm)
- Databases
- Compilers (Syntax Tree)
- Priority Queue
- AI (Decision Tree, Random forest)
- Indexing multi-dimensional information

Types of Tree

- Binary Tree
- Binary Search Tree
- AVL Tree
- B-Tree
- R-Tree
- Interval Tree
- Heap Tree

ex)



1) 5 → 15 → 8 → 7 → 9

From top left to right

2) 7 → 9 → 15 → 8 → 5

From bottom left to right

- Inorder traversal → left-subtree, node, right-subtree
- preorder traversal → node, left-subtree, right-subtree
- postorder traversal → Left, right, node

Inorder traversal

7 → 15 → 9 → 5 → 8

preorder traversal

5 → 15 → 7 → 9 → 8

postorder traversal

7 → 9 → 15 → 8 → 5

Codes

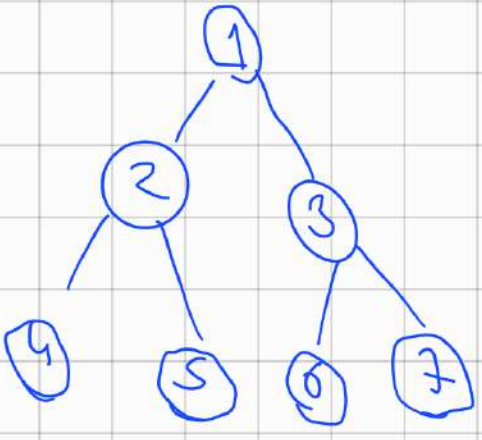
```
struct node{
int data;
struct node * left;
struct node * right;
}
```

```
void inOrderTraversal(struct node *root){
if(root==NULL)
return;
inOrderTraversal(root->left);
printf("%d ", root->data);
inOrderTraversal(root->right);
}
```

```
void preOrderTraversal(struct node *root){
if(root==NULL) return;
printf("%d ", root->data);
preOrderTraversal(root->left)
preOrderTraversal(root->right)
```

```
}

void postOrderTraversal(struct node *root) {

    if(root == NULL) return;

    postOrderTraversal(root->left);
    postOrderTraversal(root->right);
    printf("%d", root->data);
```

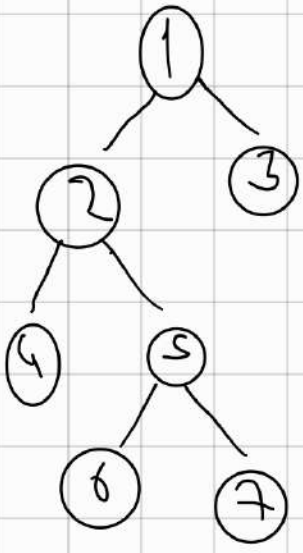
Perfect Binary Tree



it's type of binary tree which every internal nodes has exactly two child nodes and all the leaf nodes are at the same level.

Not: eğer sola yaslıysa complete binary tree

Full Binary Tree

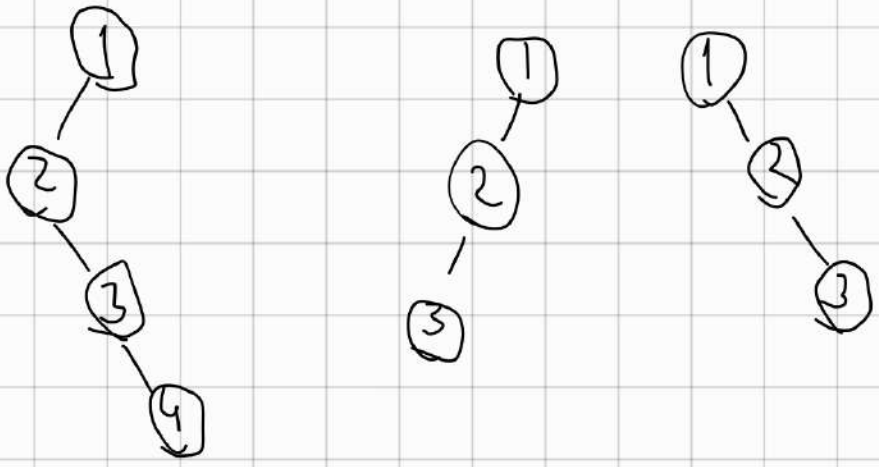


2 veya 0 çocuğu varsa.

Balanced Binary Tree

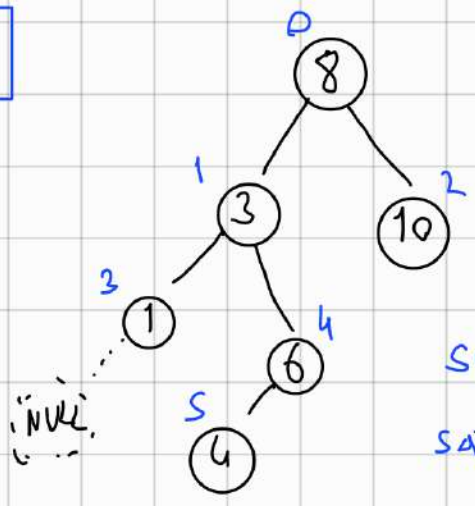
Sol ve sağ dalların yükseklik farkı en fazla 1 olmalı

Degenerate OR Pathological Tree



Binary Search Tree

8	3	10	1	6	4

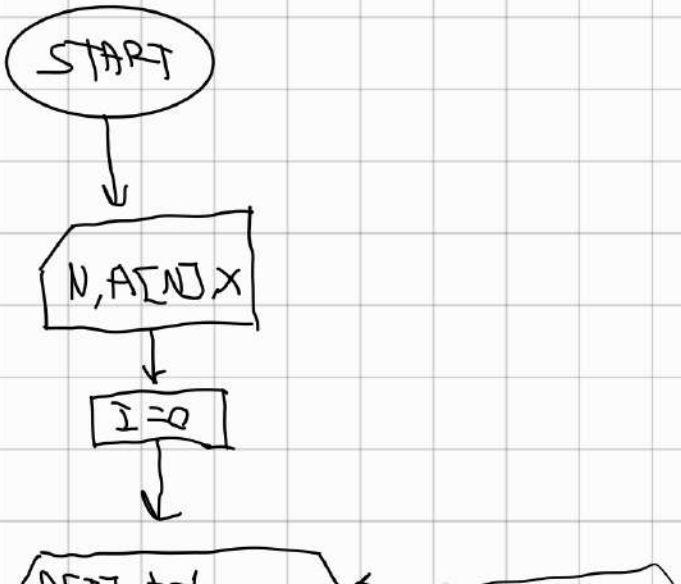


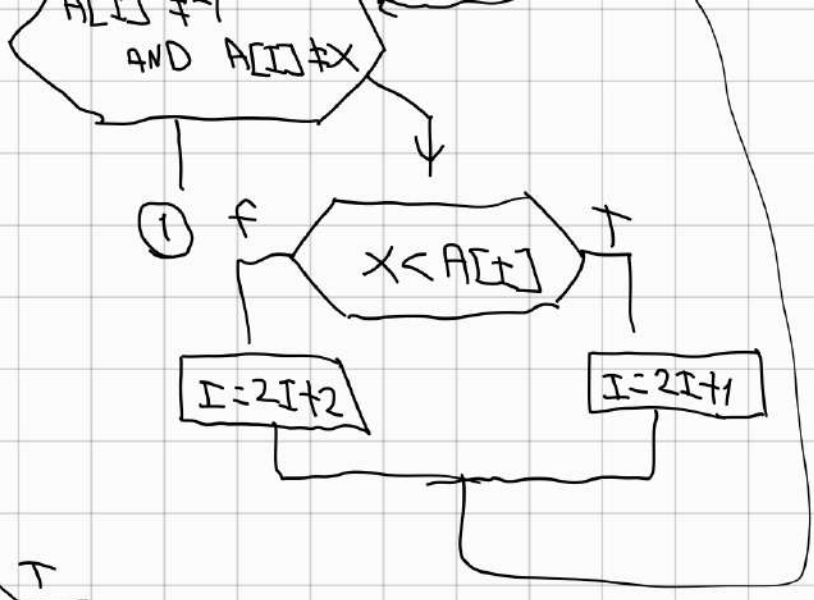
başlangıç

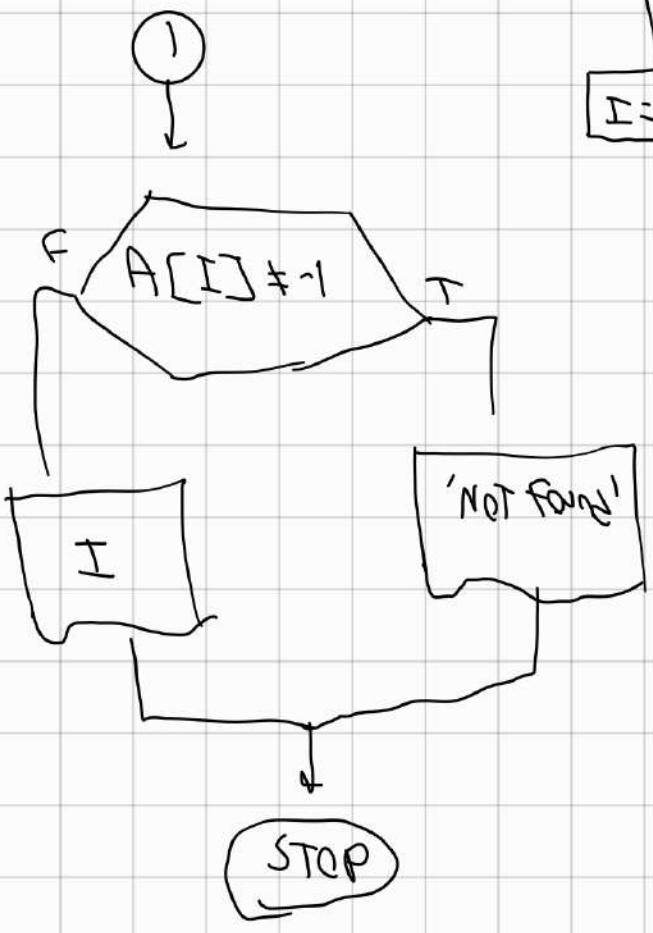
	0. index	1. index
sol	2k+1	2k
sağ	2k+2	2k+1

- Search
- Insert
- Delete

Search



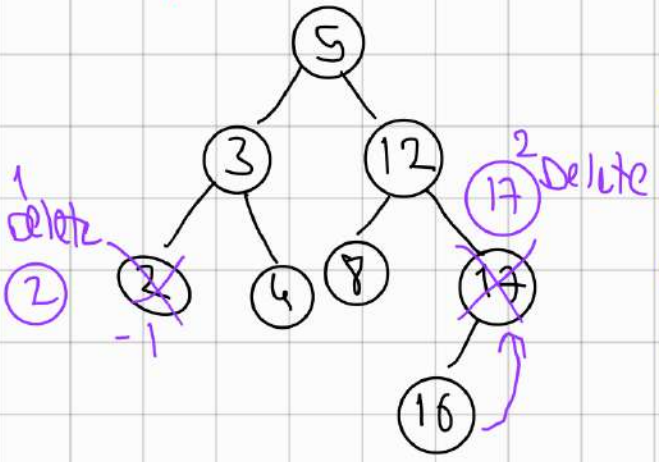




NOT: Sıralı dizide dizinin 2'ye bölüp ortası almak mantıklı çözümdür.
Sol orta, Sağ orta sonra böl

NOT: Insert ile search birbirine benzer

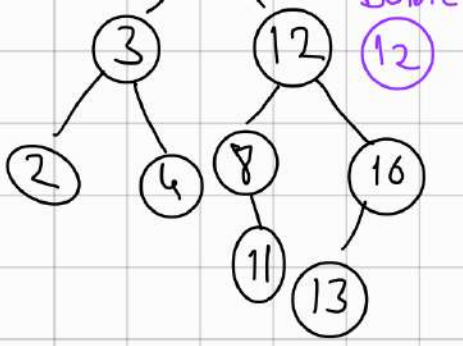
Delete



Case 1: Delete Leaf node
Case 2: Delete Internal node, with a single child
Case 3: Delete Internal node who have right-left child
Case 4: Not Found

Case 3:

1-) Get the inorder successor of that node
2-) Replace the node with the inorder successor



3) Remove the inorder successor from its original position

Inorder successor: a node is the next node in order Traversal of the Binary Tree.

Inorder Successor of an input node can also be defined as the node with the smallest key greater than the key of the input node

1. If the right subtree of a node is not NULL, then Successor lies in the right subtree.

2. If " " " " is NULL, start from root, travel down the tree, if a node's data is greater than root's data then go right side, otherwise go left side

Not: Silinecek elemanın sağında eleman varsa sağın en küçük elemanı alınır, silinecek yere konur. Eğer sağda eleman yoksa 2.yi oku.

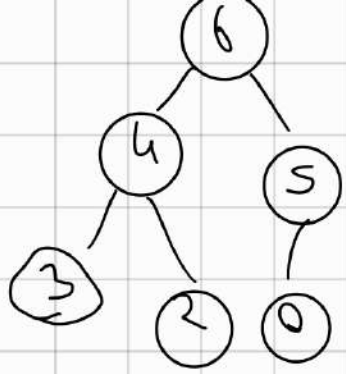
Code

```
struct node *deleteNode(struct node *root, int key){

  if(root==NULL) return root;

  if(key<root->key)
    root->left=deleteNode(root->left, key);
  if(key>root->key)
  root->right=deleteNode(root->right, key);
  else{
  if(root->left==NULL){
    struct node *tmp=root->right;
```

```
        free(root);
        return tmp;
    } else if(root->right==NULL){
        struct node *tmp=root->left;
        free(root);
        return tmp;
    }
    Struct node *tmp= minValueNode(root->right);
    root->key= tmp->key;
    root->right=deleteNode(root->right, tmp->key);
    }
    return root;
}

Struct node *minValueNode(struct node *node){
    Struct node *current=node;
    while(current && current->left!=NULL)
        current = current->left;
    return current;
}
```

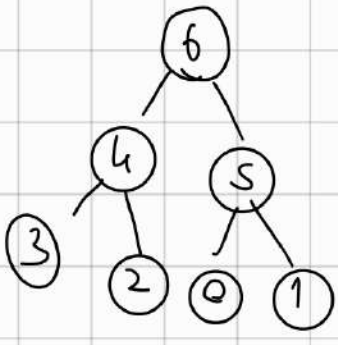
Heap Tree

- Complete Binary Tree
- Max-Heap $A[Parent(i)] \geq A[i]$
- Min-Heap $A[Parent(i)] \leq A[i]$



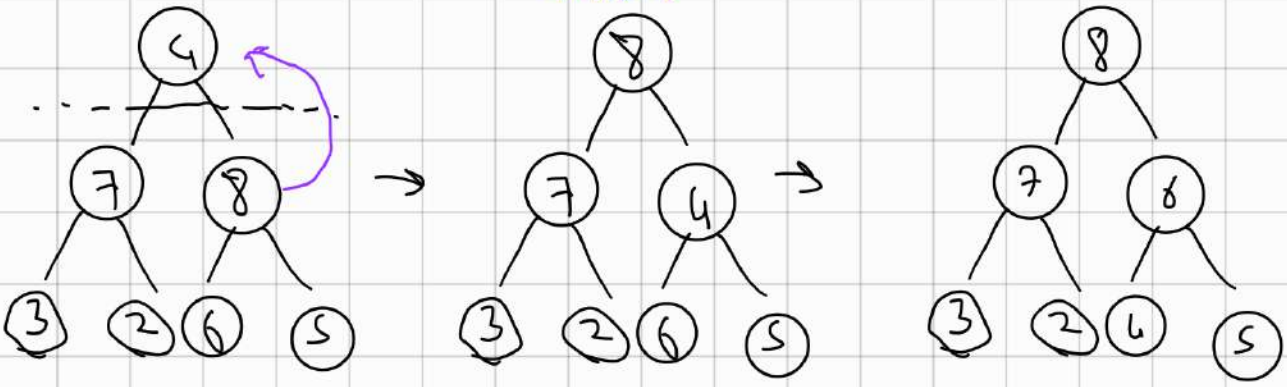
Array:

6	4	5	3	2	0	1



Aynı zamanda complete Binary Tree

Heapfy

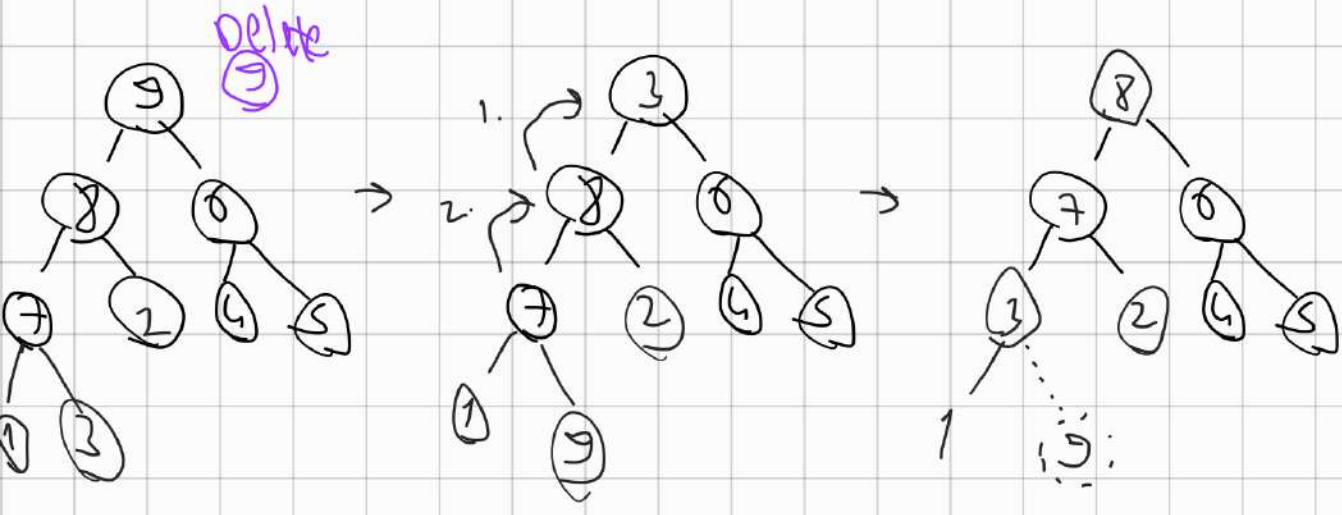


- Insert → complexity log(n)
- Delete
- search

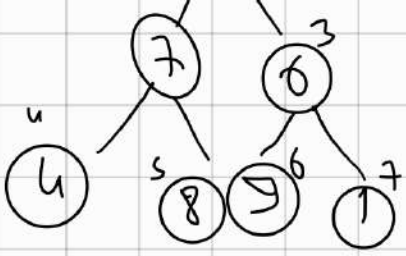
Delete operation

Silinecek elemanı en sona atarız. Sonra sileriz ve tekrar heapfy yaparız.

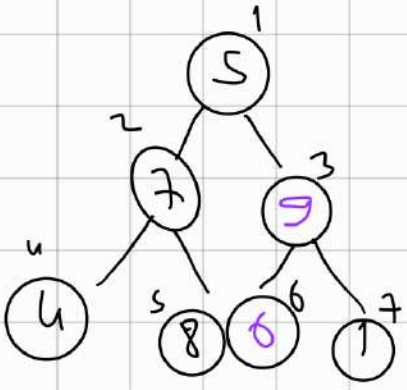
Delete (9)



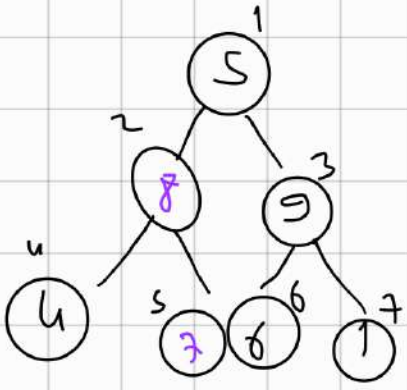
Max Heap-Tree Conversion



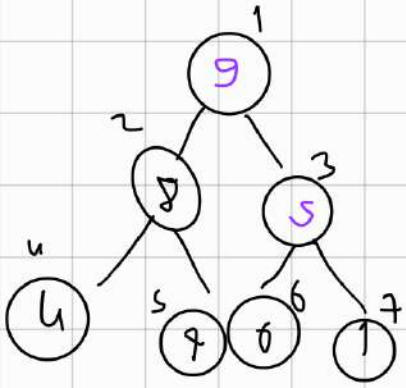
step 1



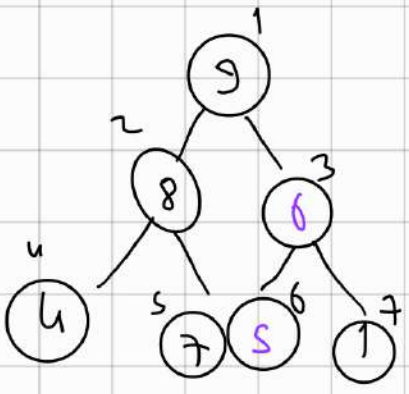
step 2



Step 3



step 4



Code

```
void buildMaxHeap(int dizi[]){
  int i;

  for(i=N/2; i>=1; i--)
    maxHeapify(dizi, i, N);
}

void maxHeapify(int dizi[], int i, int N){
  int largest;
  int left=2*i;
  int right=2*(i+1);
```

```
if((left<=N) && dizi[left] > dizi[i])

    largest=left;
  else
   largest=i;
if((right<=N) && dizi[right] > dizi[largest])

   largest=right;

  if(largest!=i){

   swap(dizi[i], dizi[largest]);
    maxHeapify(dizi, largest, N);
  }
}
```

Heap Sort

Heapify edilmiş ağaçta en büyük eleman en sona eklenir ve tekrar heapify edilir. Daha sonra N bir azaltılır. Diğeri devam eder.