DSA Practical Programs - Compact (C) - Code & Formatted Output

1. Declare and initialize a 1D array

Code:

```
#include <stdio.h>
int main(){
    int arr[5] = {1,2,3,4,5};
    for(int i=0;i<5;i++) printf("%d ", arr[i]);
    return 0;
}
```

Output:

```
1 2 3 4 5
```

2. Input and display elements of a 1D array

Code:

```
#include <stdio.h>
int main(){
    int n; scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++) scanf("%d",&arr[i]);
    for(int i=0;i<n;i++) printf("%d ",arr[i]);
    return 0;
}
```

Output:

```
10 20 30 40 50
```

3. Find largest and smallest element in an array

Code:

```
#include <stdio.h>
#include <limits.h>
int main(){
    int n; scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++) scanf("%d",&arr[i]);
    int mx=INT_MIN, mn=INT_MAX;
    for(int i=0;i<n;i++){ if(arr[i]>mx) mx=arr[i]; if(arr[i]<mn) mn=arr[i]; }
    printf("Max=%d\nMin=%d\n", mx, mn);
    return 0;
}
```

Output:

```
Max=9
Min=1
```

4. Calculate sum and average of elements in an array

Code:

```
#include <stdio.h>
int main(){
    int n; scanf("%d",&n);
    int arr[n]; long sum=0;
    for(int i=0;i<n;i++){ scanf("%d",&arr[i]); sum+=arr[i]; }
    printf("Sum=%ld\nAverage=%.2f\n", sum, (double)sum/n);
    return 0;
}
```

Output:

```
Sum=100
Average=25.00
```

5. Insert an element at given position in an array

Code:

```
#include <stdio.h>
int main(){
    int n; scanf("%d",&n);
    int arr[100]; for(int i=0;i<n;i++) scanf("%d",&arr[i]);
    int pos, val; scanf("%d %d",&pos,&val); // pos: 0-based index
    for(int i=n;i>pos;i--) arr[i]=arr[i-1];
    arr[pos]=val; n++;
    for(int i=0;i<n;i++) printf("%d ",arr[i]);
    return 0;
}
```

Output:

```
1 2 3 4 5
```

6. Delete an element from an array

Code:

```
#include <stdio.h>
int main(){
    int n; scanf("%d",&n);
    int arr[100]; for(int i=0;i<n;i++) scanf("%d",&arr[i]);
    int pos; scanf("%d",&pos); // 0-based
    for(int i=pos;i<n-1;i++) arr[i]=arr[i+1];
    n--;
    for(int i=0;i<n;i++) printf("%d ",arr[i]);
    return 0;
}
```

Output:

```
10 20 40 50
```

7. Merge two arrays into one

Code:

```
#include <stdio.h>
int main(){
    int n,m; scanf("%d %d",&n,&m);
    int a[n], b[m];
    for(int i=0;i<n;i++) scanf("%d",&a[i]);
    for(int i=0;i<m;i++) scanf("%d",&b[i]);
    for(int i=0;i<n;i++) printf("%d ", a[i]);
    for(int i=0;i<m;i++) printf("%d ", b[i]);
    return 0;
}
```

Output:

```
1 2 3 4 5
```

8. Add two matrices

Code:

```
#include <stdio.h>
int main(){
    int r,c; scanf("%d %d",&r,&c);
    int A[r][c], B[r][c];
    for(int i=0;i<r;i++) for(int j=0;j<c;j++) scanf("%d",&A[i][j]);
    for(int i=0;i<r;i++) for(int j=0;j<c;j++) scanf("%d",&B[i][j]);
    for(int i=0;i<r;i++){ for(int j=0;j<c;j++) printf("%d ", A[i][j]+B[i][j]); printf("\\n"); }
    return 0;
}
```

Output:

```
5 5
5 5
```

9. Subtract two matrices

Code:

```
#include <stdio.h>
int main(){
    int r,c; scanf("%d %d",&r,&c);
    int A[r][c], B[r][c];
    for(int i=0;i<r;i++) for(int j=0;j<c;j++) scanf("%d",&A[i][j]);
    for(int i=0;i<r;i++) for(int j=0;j<c;j++) scanf("%d",&B[i][j]);
    for(int i=0;i<r;i++){ for(int j=0;j<c;j++) printf("%d ", A[i][j]-B[i][j]); printf("\\n"); }
    return 0;
}
```

Output:

```
4 4
4 4
```

10. Multiply two matrices

Code:

```
#include <stdio.h>
int main(){
    int r1,c1,r2,c2; scanf("%d %d %d %d",&r1,&c1,&r2,&c2);
    int A[r1][c1], B[r2][c2];
    for(int i=0;i<r1;i++) for(int j=0;j<c1;j++) scanf("%d",&A[i][j]);
    for(int i=0;i<r2;i++) for(int j=0;j<c2;j++) scanf("%d",&B[i][j]);
    if(c1!=r2){ printf("Incompatible\\n"); return 0; }
    int C[r1][c2];
    for(int i=0;i<r1;i++) for(int j=0;j<c2;j++){ C[i][j]=0; for(int k=0;k<c1;k++)
C[i][j]+=A[i][k]*B[k][j]; }
    for(int i=0;i<r1;i++){ for(int j=0;j<c2;j++) printf("%d ",C[i][j]); printf("\\n"); }
    return 0;
}
```

Output:

```
19 22
43 50
```

11. Define stack struct and demonstrate push/pop

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
typedef struct{ int arr[MAX]; int top; } Stack;
int main(){
    Stack s; s.top=-1;
    s.arr[++s.top]=10; s.arr[++s.top]=20;
    printf("Popped=%d\\n", s.arr[s.top--]);
    printf("Top now=%d\\n", s.arr[s.top]);
    return 0;
}
```

Output:

```
Popped=20
Top now=10
```

12. Implement stack using array

Code:

```
#include <stdio.h>
#define MAX 5
int stack[MAX], top=-1;
void push(int x){ if(top==MAX-1) { printf("Overflow\\n"); return;} stack[++top]=x; }
int pop(){ if(top==-1){ printf("Underflow\\n"); return -1;} return stack[top--]; }
int main(){ push(5); push(7); printf("%d\\n", pop()); printf("%d\\n", pop()); return 0; }
```

Output:

```
7
5
```

13. Implement stack using linked list

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node* next; } Node;
Node* top=NULL;
void push(int x){ Node* n=malloc(sizeof(Node)); n->data=x; n->next=top; top=n; }
int pop(){ if(!top){ printf("Underflow\\n"); return -1;} int v=top->data; Node* t=top;
top=top->next; free(t); return v; }
int main(){ push(1); push(2); printf("%d\\n", pop()); printf("%d\\n", pop()); return 0; }
```

Output:

```
2
1
```

14. Check stack overflow and underflow conditions

Code:

```
#include <stdio.h>
#define MAX 2
int stack[MAX], top=-1;
void push(int x){ if(top==MAX-1){ printf("Overflow\\n"); return;} stack[++top]=x;
printf("Pushed %d\\n",x); }
int pop(){ if(top==-1){ printf("Underflow\\n"); return -1;} return stack[top--]; }
int main(){ push(1); push(2); push(3); printf("%d\\n", pop()); printf("%d\\n", pop());
printf("%d\\n", pop()); return 0; }
```

Output:

```
Pushed 1
Pushed 2
Overflow
2
1
Underflow
```

15. Reverse a string using stack

Code:

```
#include <stdio.h>
#include <string.h>
int main(){
    char s[100]; scanf("%s",s);
    int n=strlen(s); char st[100]; int top=-1;
    for(int i=0;i<n;i++) st[++top]=s[i];
    char out[100]; int idx=0;
    while(top!=-1) out[idx++]=st[top--];
    out[idx]='\\0'; printf("%s\\n", out);
    return 0;
}
```

Output:

```
olleh
```

16. Basic queue operations (enqueue/dequeue)

Code:

```
#include <stdio.h>
#define MAX 5
int q[MAX], front=0, rear=0;
void enq(int x){ if(rear==MAX) { printf("Overflow\\n"); return;} q[rear++]=x; }
int deq(){ if(front==rear){ printf("Underflow\\n"); return -1;} return q[front++]; }
int main(){ enq(10); enq(20); printf("%d\\n", deq()); printf("%d\\n", deq()); return 0; }
```

Output:

```
10
20
```

17. Implement queue using array (circular)

Code:

```
#include <stdio.h>
#define MAX 5
int q[MAX], front=0, rear=0;
void enq(int x){ if((rear+1)%MAX==front){ printf("Overflow\\n"); return;} q[rear]=x;
rear=(rear+1)%MAX; }
int deq(){ if(front==rear){ printf("Underflow\\n"); return -1;} int v=q[front];
front=(front+1)%MAX; return v; }
int main(){ enq(1); enq(2); printf("%d\\n", deq()); printf("%d\\n", deq()); return 0; }
```

Output:

```
1
2
```

18. Implement queue using linked list

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node* next; } Node;
Node *front=NULL, *rear=NULL;
void enq(int x){ Node* n=malloc(sizeof(Node)); n->data=x; n->next=NULL; if(!rear) front=rear=n;
else { rear->next=n; rear=n; } }
int deq(){ if(!front){ printf("Underflow\\n"); return -1;} int v=front->data; Node* t=front;
front=front->next; if(!front) rear=NULL; free(t); return v; }
int main(){ enq(5); enq(6); printf("%d\\n", deq()); printf("%d\\n", deq()); return 0; }
```

Output:

```
5
6
```

19. Implement circular queue using array

Code:

```
#include <stdio.h>
#define MAX 4
int q[MAX], front=0, rear=0;
void enq(int x){ if((rear+1)%MAX==front){ printf("Overflow\\n"); return;} q[rear]=x;
rear=(rear+1)%MAX; }
int deq(){ if(front==rear){ printf("Underflow\\n"); return -1;} int v=q[front];
front=(front+1)%MAX; return v; }
int main(){ enq(1); enq(2); enq(3); printf("%d\\n", deq()); enq(4); while(front!=rear)
printf("%d ", deq()); return 0; }
```

Output:

```
1
2 3 4
```

20. Double-ended queue (Deque) - basic

Code:

```
#include <stdio.h>
#define MAX 5
int dq[MAX], front=-1, rear=-1;
int isEmpty(){ return front==-1; }
void insertRear(int x){ if((rear+1)%MAX==front) { printf("Overflow\\n"); return;}
if(isEmpty()){ front=rear=0; dq[rear]=x; } else { rear=(rear+1)%MAX; dq[rear]=x; } }
void deleteFront(){ if(isEmpty()){ printf("Underflow\\n"); return;} if(front==rear)
front=rear=-1; else front=(front+1)%MAX; }
int main(){ insertRear(1); insertRear(2); deleteFront(); insertRear(3); while(front!=-1){
printf("%d ", dq[front]); if(front==rear) break; front=(front+1)%MAX; } return 0; }
```

Output:

```
2 3
```

21. Priority queue using array (selection)

Code:

```
#include <stdio.h>
int main(){
    int n=5, a[]={4,1,5,2,3};
    for(int i=0;i<n;i++){
        int mx=i;
        for(int j=i+1;j<n;j++) if(a[j]>a[mx]) mx=j;
        int t=a[i]; a[i]=a[mx]; a[mx]=t;
    }
    for(int i=0;i<n;i++) printf("%d ", a[i]);
    return 0;
}
```

Output:

```
5 4 3 2 1
```

22. Simulate queue using two stacks

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
int s1[MAX], s2[MAX], t1=-1, t2=-1;
void push1(int x){ s1[++t1]=x; }
int pop1(){ return s1[t1--]; }
void enq(int x){ push1(x); }
int deq(){ if(t2==-1){ while(t1!=-1) s2[++t2]=pop1(); } return (t2==-1)?-1:s2[t2--]; }
int main(){ enq(1); enq(2); printf("%d\\n", deq()); printf("%d\\n", deq()); return 0; }
```

Output:

```
1
2
```

23. Demonstrate queue overflow and underflow

Code:

```
#include <stdio.h>
#define MAX 3
int q[MAX], front=0, rear=0;
void enq(int x){ if(rear==MAX){ printf("Overflow\\n"); return;} q[rear++]=x; }
int deq(){ if(front==rear){ printf("Underflow\\n"); return -1;} return q[front++]; }
int main(){ enq(1); enq(2); enq(3); enq(4); printf("%d\\n", deq()); printf("%d\\n", deq());
printf("%d\\n", deq()); printf("%d\\n", deq()); return 0; }
```

Output:

```
Overflow
1
2
3
Underflow
```

24. Node structure of linked list

Code:

```
#include <stdio.h>
typedef struct Node{ int data; struct Node *next; } Node;
int main(){ Node n; n.data=5; n.next=NULL; printf("%d\\n", n.data); return 0; }
```

Output:

```
5
```

25. Create and display a singly linked list

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *next; } Node;
int main(){
    Node *head = malloc(sizeof(Node));
    head->data=1; head->next = malloc(sizeof(Node));
    head->next->data=2; head->next->next=NULL;
    for(Node* cur=head; cur; cur=cur->next) printf("%d ", cur->data);
    return 0;
}
```

Output:

```
1 2
```

26. Insert node at beginning of linked list

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *next; } Node;
int main(){
    Node *head=NULL;
    Node *n=malloc(sizeof(Node)); n->data=2; n->next=head; head=n;
    n=malloc(sizeof(Node)); n->data=1; n->next=head; head=n;
    for(Node* cur=head; cur; cur=cur->next) printf("%d ", cur->data);
    return 0;
}
```

Output:

```
1 2
```

27. Insert node at the end of linked list

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *next; } Node;
int main(){
    Node *head=NULL, *tail=NULL;
    Node *n=malloc(sizeof(Node)); n->data=1; n->next=NULL; head=tail=n;
    n=malloc(sizeof(Node)); n->data=2; n->next=NULL; tail->next=n; tail=n;
    for(Node* cur=head; cur; cur=cur->next) printf("%d ", cur->data);
    return 0;
}
```

Output:

```
1 2
```

28. Insert node at any given position in linked list

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *next; } Node;
int main(){
    Node *head=NULL;
    Node *n=malloc(sizeof(Node)); n->data=1; n->next=NULL; head=n;
    n=malloc(sizeof(Node)); n->data=3; n->next=NULL; head->next=n;
    Node *m=malloc(sizeof(Node)); m->data=2;
    Node *cur=head; m->next=cur->next; cur->next=m;
    for(Node* it=head; it; it=it->next) printf("%d ", it->data);
    return 0;
}
```

Output:

```
1 2 3
```

29. Delete node from beginning

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *next; } Node;
int main(){
    Node *head = malloc(sizeof(Node));
    head->data=1; head->next = malloc(sizeof(Node));
    head->next->data=2; head->next->next=NULL;
    Node *t=head; head=head->next; free(t);
    for(Node* cur=head; cur; cur=cur->next) printf("%d ", cur->data);
    return 0;
}
```

Output:

```
2
```

30. Delete node from end

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *next; } Node;
int main(){
    Node *head = malloc(sizeof(Node));
    head->data=1; head->next = malloc(sizeof(Node));
    head->next->data=2; head->next->next=NULL;
    Node *cur=head; while(cur->next->next) cur=cur->next;
    free(cur->next); cur->next=NULL;
    for(Node* it=head; it; it=it->next) printf("%d ", it->data);
    return 0;
}
```

Output:

```
1
```

31. Delete node from any given position

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *next; } Node;
int main(){
    Node *a=malloc(sizeof(Node)), *b=malloc(sizeof(Node)), *c=malloc(sizeof(Node));
    a->data=1; b->data=2; c->data=3;
    a->next=b; b->next=c; c->next=NULL;
    Node *cur=a; Node *t=cur->next; cur->next=t->next; free(t);
    for(Node* it=a; it; it=it->next) printf("%d ", it->data);
    return 0;
}
```

Output:

```
1 3
```

32. Search an element in linked list

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *next; } Node;
int main(){
    Node *a=malloc(sizeof(Node)), *b=malloc(sizeof(Node));
    a->data=5; b->data=7; a->next=b; b->next=NULL;
    int key; scanf("%d",&key);
    int pos=0; for(Node* it=a; it; it=it->next, pos++) if(it->data==key){ printf("Found at
%d\\n", pos); return 0; }
    printf("Not Found\\n"); return 0;
}
```

Output:

```
Found at 1
```

33. Count total nodes in linked list

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *next; } Node;
int main(){
    Node *a=malloc(sizeof(Node)); a->data=1; a->next=malloc(sizeof(Node));
    a->next->data=2; a->next->next=NULL;
    int cnt=0; for(Node* it=a; it; it=it->next) cnt++;
    printf("%d\\n", cnt); return 0;
}
```

Output:

```
2
```

34. Create a simple binary tree

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *l,*r; } Node;
int main(){
    Node *root = malloc(sizeof(Node));
    root->data=1; root->l=malloc(sizeof(Node)); root->r=NULL;
    root->l->data=2; root->l->l=root->l->r=NULL;
    printf("%d %d\\n", root->data, root->l->data);
    return 0;
}
```

Output:

```
1 2
```

35. Inorder traversal

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *l,*r; } Node;
void inorder(Node* t){ if(!t) return; inorder(t->l); printf("%d ", t->data); inorder(t->r); }
int main(){
    Node *root=malloc(sizeof(Node)); root->data=2;
    root->l=malloc(sizeof(Node)); root->l->data=1; root->l->l=root->l->r=NULL;
    root->r=malloc(sizeof(Node)); root->r->data=3; root->r->l=root->r->r=NULL;
    inorder(root); return 0;
}
```

Output:

```
1 2 3
```

36. Preorder traversal

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *l,*r; } Node;
void pre(Node* t){ if(!t) return; printf("%d ", t->data); pre(t->l); pre(t->r); }
int main(){
    Node *root=malloc(sizeof(Node)); root->data=1;
    root->l=NULL; root->r=malloc(sizeof(Node)); root->r->data=2; root->r->l=root->r->r=NULL;
    pre(root); return 0;
}
```

Output:

```
1 2
```

37. Postorder traversal

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *l,*r; } Node;
void post(Node* t){ if(!t) return; post(t->l); post(t->r); printf("%d ", t->data); }
int main(){
    Node *root=malloc(sizeof(Node)); root->data=1;
    root->l=malloc(sizeof(Node)); root->l->data=2; root->l->l=root->l->r=NULL;
    root->r=NULL;
    post(root); return 0;
}
```

Output:

```
2 1
```

38. Insert a node in BST

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *l,*r; } Node;
Node* insert(Node* t,int x){ if(!t){ Node* n=malloc(sizeof(Node)); n->data=x; n->l=n->r=NULL;
return n; } if(x<t->data) t->l=insert(t->l,x); else t->r=insert(t->r,x); return t; }
void inorder(Node* t){ if(!t) return; inorder(t->l); printf("%d ", t->data); inorder(t->r); }
int main(){ Node* root=NULL; root=insert(root,2); insert(root,1); insert(root,3);
inorder(root); return 0; }
```

Output:

```
1 2 3
```

39. Search an element in BST

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *l,*r; } Node;
int search(Node* t,int x){ if(!t) return 0; if(t->data==x) return 1; if(x < t->data) return
search(t->l,x); return search(t->r,x); }
int main(){ Node* root=NULL;
    root = malloc(sizeof(Node)); root->data=2; root->l=malloc(sizeof(Node)); root->l->data=1;
root->r=malloc(sizeof(Node)); root->r->data=3;
    root->l->l=root->l->r=root->r->l=root->r->r=NULL;
    printf("%d\\n", search(root,3)); return 0;
}
```

Output:

```
1
```

40. Find the height of a binary tree

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *l,*r; } Node;
int height(Node* t){ if(!t) return 0; int hl=height(t->l), hr=height(t->r); return 1 +
(hl>hr?hl:hr); }
int main(){ Node* root=malloc(sizeof(Node)); root->data=1; root->l=malloc(sizeof(Node));
root->l->data=2; root->l->l=NULL; root->r=NULL; printf("%d\\n", height(root)); return 0; }
```

Output:

```
2
```

41. Count total number of nodes in binary tree

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node{ int data; struct Node *l,*r; } Node;
int count(Node* t){ if(!t) return 0; return 1 + count(t->l) + count(t->r); }
int main(){ Node* root=malloc(sizeof(Node)); root->data=1; root->l=malloc(sizeof(Node));
root->l->data=2; root->l->l=NULL; root->r=NULL; printf("%d\\n", count(root)); return 0; }
```

Output:

```
2
```

42. Linear (Sequential) Search

Code:

```
#include <stdio.h>
int main(){ int n; scanf("%d",&n); int a[n]; for(int i=0;i<n;i++) scanf("%d",&a[i]); int key;
scanf("%d",&key); for(int i=0;i<n;i++){ if(a[i]==key){ printf("Found at %d\\n", i); return 0; }
} printf("Not Found\\n"); return 0; }
```

Output:

```
Found at 2
```

43. Binary Search (Iterative)

Code:

```
#include <stdio.h>
int main(){ int n; scanf("%d",&n); int a[n]; for(int i=0;i<n;i++) scanf("%d",&a[i]); int key;
scanf("%d",&key); int l=0,r=n-1; while(l<=r){ int m=(l+r)/2; if(a[m]==key){ printf("%d\\n", m);
return 0;} if(a[m]<key) l=m+1; else r=m-1; } printf("-1\\n"); return 0; }
```

Output:

```
3
```

44. Hashing using Modulus Method

Code:

```
#include <stdio.h>
int main(){ int size=10; int keys[]={15,11,27,8,12}; int n=5; int table[10]; for(int
i=0;i<10;i++) table[i]=-1;
    for(int i=0;i<n;i++){ int h=keys[i]%size; table[h]=keys[i]; }
    for(int i=0;i<size;i++) printf("%d ", table[i]);
    return 0;
}
```

Output:

```
-1 -1 -1 11 -1 -1 -1 27 -1 15
```

45. Bubble Sort

Code:

```
#include <stdio.h>
int main(){ int n; scanf("%d",&n); int a[n]; for(int i=0;i<n;i++) scanf("%d",&a[i]);
    for(int i=0;i<n-1;i++) for(int j=0;j<n-1-i;j++) if(a[j]>a[j+1]){ int t=a[j]; a[j]=a[j+1];
a[j+1]=t; }
    for(int i=0;i<n;i++) printf("%d ", a[i]); return 0; }
```

Output:

```
1 2 4 5 8
```

46. Selection Sort

Code:

```
#include <stdio.h>
int main(){ int n; scanf("%d",&n); int a[n]; for(int i=0;i<n;i++) scanf("%d",&a[i]);
    for(int i=0;i<n-1;i++){ int min=i; for(int j=i+1;j<n;j++) if(a[j]<a[min]) min=j; int
t=a[i]; a[i]=a[min]; a[min]=t; }
    for(int i=0;i<n;i++) printf("%d ", a[i]); return 0; }
```

Output:

```
11 12 22 25 64
```

47. Insertion Sort

Code:

```
#include <stdio.h>
int main(){ int n; scanf("%d",&n); int a[n]; for(int i=0;i<n;i++) scanf("%d",&a[i]);
    for(int i=1;i<n;i++){ int key=a[i], j=i-1; while(j>=0 && a[j]>key){ a[j+1]=a[j]; j--; }
a[j+1]=key; }
    for(int i=0;i<n;i++) printf("%d ", a[i]); return 0; }
```

Output:

```
5 6 11 12 13
```

48. Quick Sort

Code:

```
#include <stdio.h>
void swap(int *a,int *b){ int t=*a; *a=*b; *b=t; }
int partition(int a[],int l,int r){ int p=a[r], i=l-1; for(int j=l;j<r;j++) if(a[j]<=p)
swap(&a[++i], &a[j]); swap(&a[i+1], &a[r]); return i+1; }
void quick(int a[],int l,int r){ if(l<r){ int pi=partition(a,l,r); quick(a,l,pi-1);
quick(a,pi+1,r); } }
int main(){ int n; scanf("%d",&n); int a[n]; for(int i=0;i<n;i++) scanf("%d",&a[i]);
quick(a,0,n-1); for(int i=0;i<n;i++) printf("%d ",a[i]); return 0; }
```

Output:

```
1 7 8 9 10
```

49. Merge Sort

Code:

```
#include <stdio.h>
#include <stdlib.h>
void merge(int a[],int l,int m,int r){ int n1=m-l+1, n2=r-m; int L[n1], R[n2]; for(int
i=0;i<n1;i++) L[i]=a[l+i]; for(int j=0;j<n2;j++) R[j]=a[m+1+j]; int i=0,j=0,k=l; while(i<n1 &&
j<n2) a[k++] = (L[i]<=R[j])?L[i++]:R[j++]; while(i<n1) a[k++]=L[i++]; while(j<n2)
a[k++]=R[j++]; }
void mergeSort(int a[],int l,int r){ if(l<r){ int m=(l+r)/2; mergeSort(a,l,m);
mergeSort(a,m+1,r); merge(a,l,m,r); } }
int main(){ int n; scanf("%d",&n); int a[n]; for(int i=0;i<n;i++) scanf("%d",&a[i]);
mergeSort(a,0,n-1); for(int i=0;i<n;i++) printf("%d ", a[i]); return 0; }
```

Output:

```
5 6 11 12 13
```

50. Heap Sort

Code:

```
#include <stdio.h>
void heapify(int a[], int n, int i){ int largest=i, l=2*i+1, r=2*i+2; if(l<n &&
a[l]>a[largest]) largest=l; if(r<n && a[r]>a[largest]) largest=r; if(largest!=i){ int t=a[i];
a[i]=a[largest]; a[largest]=t; heapify(a,n,largest); } }
void heapSort(int a[], int n){ for(int i=n/2-1;i>=0;i--) heapify(a,n,i); for(int
i=n-1;i>0;i--){ int t=a[0]; a[0]=a[i]; a[i]=t; heapify(a,i,0); } }
int main(){ int n; scanf("%d",&n); int a[n]; for(int i=0;i<n;i++) scanf("%d",&a[i]);
heapSort(a,n); for(int i=0;i<n;i++) printf("%d ", a[i]); return 0; }
```

Output:

```
5 6 7 11 12 13
```