CS-6313-Data Structure

PRACTICAL NO.03

Dry Running of Algorithm & Analysis of Algorithm

Name _____ROLL-NO_____SEM/SEC_____

PLO	CLO	LL
4	1	C-3

Objectives

- To understand the importance of dry running in debugging and algorithm analysis.
- To learn step counting and derive time complexity.
- To compare theoretical and practical efficiency of algorithms.
- To build strong foundations for algorithm design.

Part A: Dry Running Exercises

Students are given codes and must **trace line by line** to show variables' values, loop counts, and outputs.

Dry Run Table Format:

| Step | Line Executed | Variables State | Output (if any) |

Code 1: Array Traversal

```
for(int i = 0; i < 5; i++) {
    arr[i] = i * 2;
    printf("%d ", arr[i]);
}
```

Step	Line Executed	Variables (arr[], i)	Output
1	for(int i=0; i<5; i++)	i=0, arr={0,0,0,0,0}	
2	arr[i] = i*2	i=0, arr={0,0,0,0,0}	
3	printf("%d ", arr[i])	i=0, arr={0,0,0,0,0}	0
4	i++ (loop increment)	i=1, arr={0,0,0,0,0}	0
5	arr[i] = i*2	i=1, arr={0,2,0,0,0}	0
6	printf("%d ", arr[i])	i=1, arr={0,2,0,0,0}	0 2
7	i++	i=2, arr={0,2,0,0,0}	0 2
8	arr[i] = i*2	i=2, arr={0,2,4,0,0}	0 2
9	printf("%d ", arr[i])	i=2, arr={0,2,4,0,0}	0 2 4
10	i++	i=3, arr={0,2,4,0,0}	0 2 4
11	arr[i] = i*2	i=3, arr={0,2,4,6,0}	0 2 4
12	printf("%d ", arr[i])	i=3, arr={0,2,4,6,0}	0 2 4 6
13	i++	i=4, arr={0,2,4,6,0}	0 2 4 6
14	arr[i] = i*2	i=4, arr={0,2,4,6,8}	0 2 4 6
15	printf("%d ", arr[i])	i=4, arr={0,2,4,6,8}	0 2 4 6 8
16	i++	i=5, arr={0,2,4,6,8}	0 2 4 6 8
17	for loop condition fails	i=5, arr={0,2,4,6,8}	0 2 4 6 8

Code 2: Linear Search

```
int linearSearch(int arr[], int n, int key) {
    for(int i=0; i<n; i++) {
        if(arr[i] == key)
            return i;
    }
    return -1;
}
```

Step	Line Executed	Variables (arr[], i, n, key)	Condition/Output
1	i=0; for(i=0;i<n;i++)	i=0, arr={2,5,7,9,10}, key=9	i<n → 0<5 → true
2	if(arr[i]==key)	arr[0]=2, key=9	2==9 → false
3	i++	i=1	
4	if(arr[i]==key)	arr[1]=5, key=9	5==9 → false
5	i++	i=2	
6	if(arr[i]==key)	arr[2]=7, key=9	7==9 → false
7	i++	i=3	
8	if(arr[i]==key)	arr[3]=9, key=9	9==9 → true
9	return i	i=3	Output → 3

Code 3: Binary Search

```
int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n-1;
    while(low <= high) {
        int mid = (low + high) / 2;
        if(arr[mid] == key) return mid;
        else if(arr[mid] < key) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}
```

Step	Line Executed	Variables (arr[], low, high, mid, key)	Condition / Output
1	low=0, high=n-1	low=0, high=4, key=7	low<=high → 0<=4 → true
2	mid=(low+high)/2	mid=(0+4)/2=2	
3	if(arr[mid]==key)	arr[2]=5, key=7	5==7 → false

4	else if(arr[mid]<key)	5<7 → true	low=mid+1 → low=3
5	while(low<=high)	low=3, high=4	3<=4 → true
6	mid=(low+high)/2	mid=(3+4)/2=3	
7	if(arr[mid]==key)	arr[3]=7, key=7	7==7 → true
8	return mid	mid=3	Output → 3

Array Insertion at Beginning, Middle, and End

Q1. Array Insertion at Beginning

```c
#include <stdio.h>
int main() {
    int arr[6] = {10, 20, 30, 40, 50};
    int n = 5;
    int pos = 0, val = 5;   // Insert at beginning
    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;
}
```

Step	Line Executed	Variables (arr[], i, n, pos, val, etc.)	Output (if any)
1			
2			
3			
4			
5			
…			

Final Output: _____

Q2. Array Insertion at Middle

```c
#include <stdio.h>
int main() {
    int arr[6] = {1, 2, 4, 5, 6};
    int n = 5;
    int pos = 2, val = 3;  // Insert 3 at index 2
    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;
}
```

Step	Line Executed	Variables (arr[], i, n, pos, val)	Output (if any)
1			
2			
3			
4			
5			
…			

Final Output: _____

Q3. Array Insertion at End

```c
#include <stdio.h>
int main() {
    int arr[6] = {2, 4, 6, 8, 10};
    int n = 5;
    int val = 12;
    arr[n] = val;    // insert at end
    n++;
    for(int i=0; i<n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

Step	Line Executed	Variables (arr[], i, n, val)	Output (if any)
1			
2			
3			
4			
5			
…			

Final Output: _____

Q4. Array Deletion

```c
#include <stdio.h>
int main() {
    int arr[6] = {2, 4, 6, 8, 10};
    int n = 5;
    int pos = 2;   // delete element at index 2
    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;
}
```

Step	Line Executed	Variables (arr[], i, n, pos)	Output (if any)
1			
2			
3			
4			
5			
…			

Final Output: _____

Q5. Summing Elements of Array

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

Step	Line Executed	Variables (arr[], i, sum)	Output (if any)
1			
2			
3			
4			
5			
…			

Final Output: _____

Q6. Counting Even and Odd Numbers

```c
#include <stdio.h>
int main() {
    int arr[6] = {1, 4, 7, 10, 13, 16};
    int even = 0, odd = 0;
    for(int i=0; i<6; i++) {
        if(arr[i] % 2 == 0)
            even++;
        else
            odd++;
    }
    printf("Even=%d Odd=%d", even, odd);
    return 0;
}
```

Step	Line Executed	Variables (arr[], i, even, odd)	Output (if any)
1			
2			
3			
4			
5			
…			

Final Output: _____

Part B – Searching Algorithms

1) Linear Search

```c
#include <stdio.h>
int linearSearch(int arr[], int n, int key) {
    for(int i=0; i<n; i++) {
        if(arr[i] == key)
            return i;
    }
    return -1;
}
int main() {
    int arr[5] = {2, 5, 7, 9, 10};
    int key = 9;
    int result = linearSearch(arr, 5, key);
    printf("%d", result);
    return 0;
}
```

Dry Run Table (Key = 9)

Step	Line Executed	Variables (i, key, arr[i])	Condition (`arr[i] == key`)	Output
1	for(i=0; i<5; i++)	i=0, arr[0]=2, key=9	2==9 → false	
2	for(i=1; i<5; i++)	i=1, arr[1]=5, key=9	5==9 → false	
3	for(i=2; i<5; i++)	i=2, arr[2]=7, key=9	7==9 → false	
4	for(i=3; i<5; i++)	i=3, arr[3]=9, key=9	9==9 → true	return 3

Dry Run Table (Key = 11)

Step	Line Executed	Variables (i, key, arr[i])	Condition (`arr[i] == key`)	Output
1				
2				
3				
4				

Dry Running (any random number)

Step	Line Executed	Variables (i, key, arr[i])	Condition (`arr[i] == key`)	Output
1				
2				
3				
4				

2) Binary Search

```c
#include <stdio.h>
int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n-1;
    while(low <= high) {
        int mid = (low + high) / 2;
        if(arr[mid] == key) return mid;
        else if(arr[mid] < key) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}
int main() {
    int arr[5] = {1, 3, 5, 7, 9};
    int key = 7;
    int result = binarySearch(arr, 5, key);
    printf("%d", result);
    return 0;
}
```

Dry Run Table (Key = 7)
Array: 1, 3,
5, 7, 9}, n = 5, key = 7

Step	Line Executed	low	high	mid	arr[mid]	Condition Checked	Action
1	while(low<=high)	0	4	—	—	0<=4 → true	continue loop
2	mid=(low+high)/2	0	4	2	5	—	mid=2
3	if(arr[mid]==key)	—	—	2	5	5==7 → false	skip
4	else if(arr[mid]<key)	—	—	2	5	5<7 → true	low=mid+1=3
5	while(low<=high)	3	4	—	—	3<=4 → true	continue loop
6	mid=(low+high)/2	3	4	3	7	—	mid=3
7	if(arr[mid]==key)	—	—	3	7	7==7 → true	return 3

Task 1: Dry Run (Key = 9)
Fill in the table below step by step:

Step	Line Executed	low	high	mid	arr[mid]	Condition Checked	Action
1							
2							
…							

Final Output: _____

Part C – Analysis of Algorithm

Section 1 – Example with Solution

Example 1: Simple Loop

```
for(int i=0; i<n; i++) {
    printf("%d", i);
}
```

Step Counting (n=5):

- Initialization: 1 step
- Condition check: n+1 steps
- Increment: n steps
- Print statement: n steps

Total Steps = `1 + (n+1) + n + n = 3n + 2`

Formula: T(n) = 3n + 2

Big-O Complexity: O(n)

Section 2 – Practice Tasks (Without Solutions)

Task 1: Single Loop

```
for(int i=0; i<n; i++) {
    sum += i;
}
```

n	Steps (manual count)	Formula T(n)	Big-O Complexity
5	1 + 6 + 5 + 5 = 17	3n + 2 = 17	O(n)
10			
20			

Task 2: Nested Loop (Square Complexity)

```
for(int i=0; i<n; i++) {
    for(int j=0; j<n; j++) {
        printf("%d", i+j);
    }
}
```

n	Steps (manual count)	Formula T(n)	Big-O Complexity
2			
3			
4			

Task 3: Triangular Loop (Sum of Series)

```
for(int i=0; i<n; i++) {
    for(int j=0; j<i; j++) {
        printf("*");
    }
}
```

n	Steps (manual count)	Formula T(n)	Big-O Complexity
3			
5			
n			

Task 4: Binary Search (Logarithmic Complexity)

```
int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n-1;
    while(low <= high) {
        int mid = (low+high)/2;
        if(arr[mid] == key) return mid;
        else if(arr[mid] < key) low = mid+1;
        else high = mid-1;
    }
    return -1;
}
```

n	Key Position	Comparisons (steps)	Complexity
5	Found (middle)		
5	Not Found		
7	Found (last)		

Code for Student Analysis

```
#include <stdio.h>

int main() {
    int n = 10, sum = 0;

    // Single loop
    for(int i = 0; i < n; i++) {
        sum += i;   // statement 1
    }

    // Nested loop
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            sum += (i + j);   // statement 2
        }
    }

    // Conditional loop
    for(int i = 0; i < n; i++) {
        if(i % 2 == 0) {
            sum += i;   // statement 3
        } else {
            sum -= i;   // statement 4
        }
    }

    printf("Final Sum = %d\n", sum);
    return 0;
}
```

Task 1 – Single Loop

1. Count the number of steps.
2. Write formula T(n).
3. Find Big-O.

Task 2 – Nested Loop

1. Count steps for outer loop, inner loop, and body.
2. Write T(n).
3. Find Big-O.

Task 3 – Conditional Loop

1. Count steps for `if` and `else`.
2. Write T(n).
3. Find Big-O.

Task 4 – Final Program Complexity

1. Combine T(n) for all 3 parts.
2. Simplify into a single formula.
3. Find overall Big-O.

Solution Table – Execution Steps & Complexity

Part of Code	Execution Steps Breakdown	Formula T(n)	Big-O Complexity
Part 1: Single Loop			
Part 2: Nested Loop			
Part 3: Conditional Loop			
Final Program			