Manual

Sudarshan Dodiya

May 15, 2020

Contents

1 Permutation and Combination

1.1 Sample I/O

Inputs	Permutation	Combination	Total Time Taken
n r	nPr	nCr	
1 1	1	1	.58x10^{-4} sec
8 4	1680	70	.52x10^{-4} sec
5 5	120	1	.74x10^{-4} sec
12 10	238500800	66	.6x10^{-4} sec

1.2 Analysis

$$^{n}C_{r} = \frac{n!}{(n-r)!r!} \tag{1}$$

$$^{n}P_{r} = \frac{n!}{(n-r)!} \tag{2}$$

1.3 Algorithm

1. Start
2. Enter the Values of n and r
3. if n>r print "Not Possible" exit(0)
4. Perform the Permutation and Combination operations.
5. Print "Permutations and Combinations are %d and %d"
6. Stop

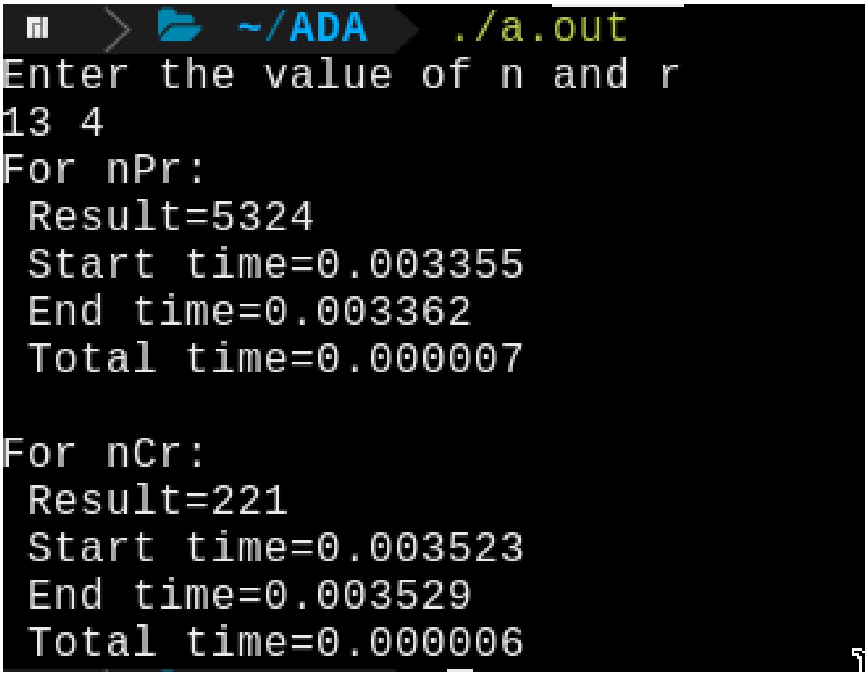
1.4 Code

```c
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
int fact(int n){
    int i;
    int fact=1;
    for (i=1;i<=n;i++)
        fact = fact*i;
    return fact;
```

```
        }

void main(){
    int n,r,P,C;
    printf("Enter the value of n and r\n");
    scanf("%d %d",&n,&r);
    double ss,ee,tt;
    clock_t st,et;
    st=clock();
    P=fact(n)/fact(n-r);
    et=clock();
    ss=((double)st/CLOCKS_PER_SEC);
    ee=((double)et/CLOCKS_PER_SEC);
    tt=((double)(et-st)/CLOCKS_PER_SEC);
    printf("For nPr: \n Result=%d \n Start time=%lf\n End time=%lf \n Total time=%lf",
    st=clock();
    C=fact(n)/(fact(n-r)*fact(r));
    et=clock();
    ss=((double)st/CLOCKS_PER_SEC);
    ee=((double)et/CLOCKS_PER_SEC);
    tt=((double)(et-st)/CLOCKS_PER_SEC);
    printf("\n\nFor nCr: \n Result=%d \n Start time=%lf \n End time=%lf \n Total time=
}
```

1.5 Output



```
~/ADA  ./a.out
Enter the value of n and r
13 4
For nPr:
 Result=5324
 Start time=0.003355
 End time=0.003362
 Total time=0.000007

For nCr:
 Result=221
 Start time=0.003523
 End time=0.003529
 Total time=0.000006
```

1.6 Time Complexity Analysis

2 Linear Search and Binary Search

2.1 Sample I/O

Data set	Input key	Time taken for Linear Search	Time taken for Binary search
1 2 3 4	2	$5.9 x 10^{-5}$	$.9 x 10^{-5}$
1 2 3 4	6	$1.3 x 10^{-5}$	$.9 x 10^{-5}$
1 2 3 4 5 6 7 8	7	$7.8 x 10^{-5}$	$1.2 x 10^{-5}$
1 2 3 4 5 6 7 8	22	$1.3 x 10^{-5}$	$.8 x 10^{-5}$

2.2 Analysis

2.2.1 Linear Search

```
LS(int a[n]){
    int f=0;
    for (i=0;i<n;i++){
        if (a[i]==key){
            f=1;
            break;
        }
    }
    if (f==0)
        printf("Key not found");
    else
        printf("Key Found at %d",i+1);
}
```

2.2.2 Binary Search

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

```
        return 1;
}
```

2.3 Algorithm

2.3.1 Linear Search

1. Start

2. Read size and array

   ```
   scanf("%d",&n);
   for (i=0;i<n;i++)
       scanf("%d",a[i]);
   ```

3. Read key element

4. Implement linear search function LS.

5. Print results

6. Stop

2.3.2 Binary Search

1. Start

2. Read size and array

   ```
   scanf("%d",&n);
   for (i=0;i<n;i++)
       scanf("%d",a[i]);
   ```

3. Read key element to be searched.

4. Implement Binary search function BS.

5. if the function return 1, then the element is not found, go to 7

6. print results

7. Stop

2.4 Code

2.4.1 Linear Search

```c
#include<stdio.h>
#include<time.h>
int a[50];
void LS(int n, int key){
    int f=0,i;

    for (i=0;i<n;i++){
        if (a[i]==key){
            f=1;
            break;
        }
    }
    if (f==0)
        printf("Key not found");
    else
        printf("Key Found at %d\n",i+1);
}
void main(){
    clock_t st,et;
    double ss,ee;
    int n,key,i;
    printf("Enter number of elements:");
    scanf("%d",&n);
    printf("Enter array:\n");
    for (i=0;i<n;i++)
        scanf("%d",&a[i]);
    printf("Enter the key element:");
    scanf("%d",&key);
    st=clock();
    LS(n,key);
    et=clock();
    ss=((double)st/CLOCKS_PER_SEC);
    ee=((double)et/CLOCKS_PER_SEC);
    printf(" Start time:%lf \n End time:%lf \n Total time:%lf \n",ss,ee,ee-ss);
}
```

2.4.2 Binary Search

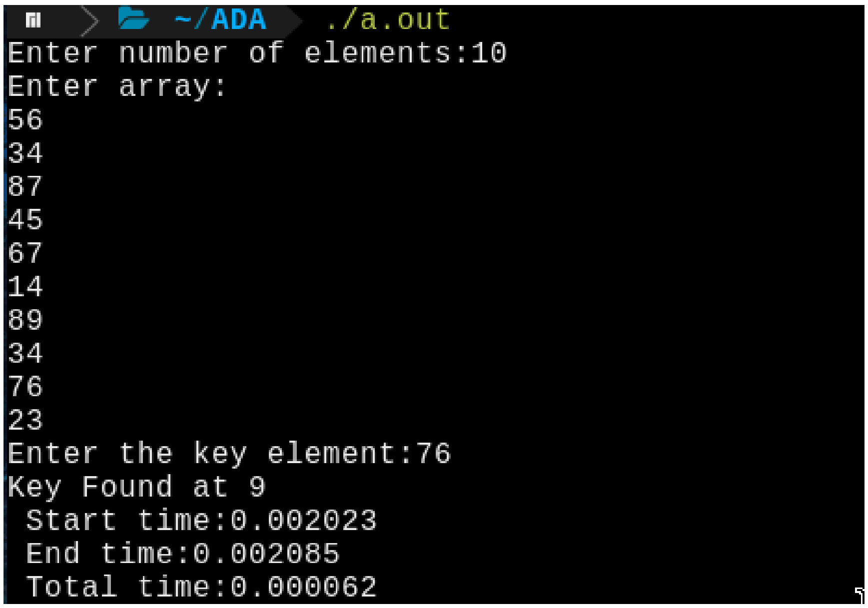
```c
#include <stdio.h>
#include <time.h>
int a[50];
int BS(int n,int key){
    int low=0,high=n,mid;
    while(low<high){
        mid=(low+high)/2;
        if (a[mid]==key)
            return mid;
        else if (a[mid]<key)
            low=mid+1;
        else if (a[mid]>key)
            high=mid-1;
        else
            return 99;
    }
}

void main(){
    clock_t st,et;
    double ss,ee;
    int n,key,i;
    printf("Enter number of elements:");
    scanf("%d",&n);
    printf("Enter array:\n");
    for (i=0;i<n;i++)
        scanf("%d",&a[i]);
    printf("Enter the key element:");
    scanf("%d",&key);
    st=clock();
    int x=BS(n,key);
    et=clock();
    ss=((double)st/CLOCKS_PER_SEC);
    ee=((double)et/CLOCKS_PER_SEC);
    if (x==99)
        printf(" Not found");
    else
        printf(" Found at: %d position\n",x+1);
```

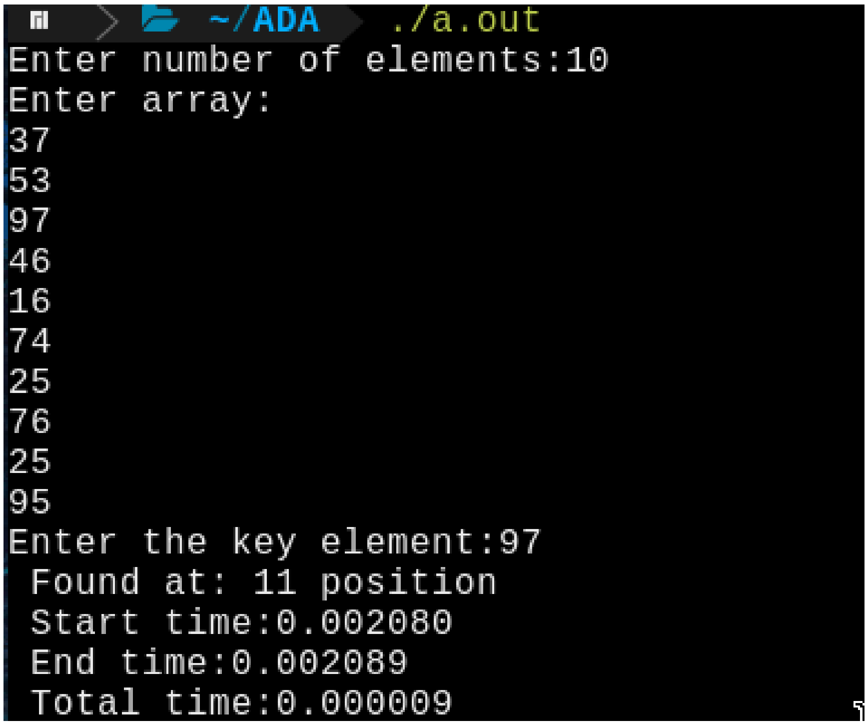
```
    printf(" Start time:%lf \n End time:%lf \n Total time:%lf \n",x,ss,ee,ee-ss);
}
```

2.5 Output

2.5.1 Linear Search



2.5.2 Binary Search



2.6 Time Complexity Analysis

3 Tower of Hanoi

3.1 Sample I/O

Number of disks	Time taken
3	1.94×10^{-4}
4	3.4×10^{-4}
5	5.8×10^{-4}
6	12.07×10^{-4}

3.2 Analysis

```
void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod)
{
    if (n == 1)
    {
        printf("\n Move disk 1 from rod %c to rod %c", from_rod, to_rod);
        return;
    }
    towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
    printf("\n Move disk %d from rod %c to rod %c", n, from_rod, to_rod);
    towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
}
```

3.3 Algorithm

3.4 Code

```
#include<stdio.h>
#include<time.h>
void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod)
{
    if (n == 1)
    {
        printf("\n Move disk 1 from rod %c to rod %c", from_rod, to_rod);
        return;
    }
    towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
    printf("\n Move disk %d from rod %c to rod %c", n, from_rod, to_rod);
    towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
}
```

```c
int main()
{
    clock_t st,et;
    double ss,ee;
    int n = 4;
    st=clock();
    towerOfHanoi(n, 'A', 'C', 'B');
    et=clock();
    ss=((double)st/CLOCKS_PER_SEC);
    ee=((double)et/CLOCKS_PER_SEC);
    printf(" Start time=%lf \n End time=%lf \n Total time=%lf \n ",ss,ee,ee-ss);
    return 0;
}
```

3.5 Output

```
~/ADA ./a.out
Number of disks:4

 Move disk 1 from rod A to rod B
 Move disk 2 from rod A to rod C
 Move disk 1 from rod B to rod C
 Move disk 3 from rod A to rod B
 Move disk 1 from rod C to rod A
 Move disk 2 from rod C to rod B
 Move disk 1 from rod A to rod B
 Move disk 4 from rod A to rod C
 Move disk 1 from rod B to rod C
 Move disk 2 from rod B to rod A
 Move disk 1 from rod C to rod A
 Move disk 3 from rod B to rod C
 Move disk 1 from rod A to rod B
 Move disk 2 from rod A to rod C
 Move disk 1 from rod B to rod C
 Start time=0.003481
 End time=0.003832
 Total time=0.000351
```

3.6 Time Complexity Analysis

T(n)=O($2^{(n+1)}$ -1), or you can say O(2^n).

4 Quick Sort

4.1 Sample I/O

Data set	Time taken
1 2 3 4	$.1 \times 10^{-4}$
4 3 2 1	$.13 \times 10^{-4}$
1 2 3 4 5 6 7 8	$.12 \times 10^{-4}$
8 7 6 5 4 3 2 1	$.12 \times 10^{-4}$

4.2 Analysis

```
int partition (int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high- 1; j++)
    {
        if (arr[j] < pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

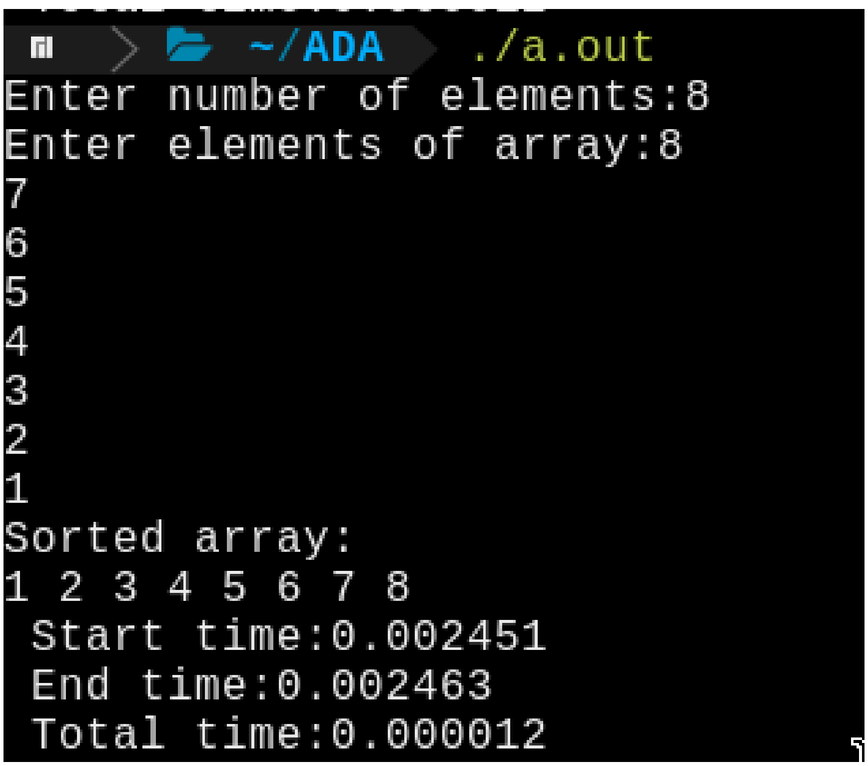
4.3 Algorithm

1. Start
2. Read the number of elements of the array.
3. Read the elements of the array.

```
scanf("%d",&n);
for (int i=0;i<n;i++){
    scanf("%d",&arr[i]);
}
```

4. Perform quicksort operation.
5. Print the resulting array.
6. Stop.

4.4 Output



4.5 Time Complexity Analysis

5 Merge Sort

5.1 Sample I/O

Data set	Time taken
1 2 3 4	$.11x10^{-4}$
4 3 2 1	$.10x10^{-4}$
1 2 3 4 5 6 7 8	$.14x10^{-4}$
8 7 6 5 4 3 2 1	$.17x10^{-4}$

5.2 Analysis

```
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1)
    {
```

```
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = l+(r-l)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
```

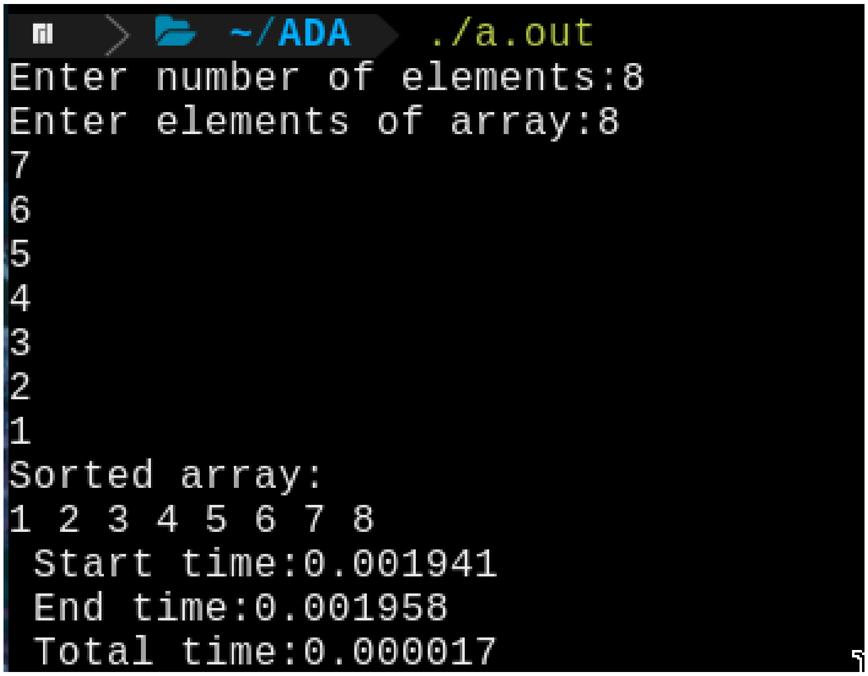
5.3 Algorithm

1. Start
2. Read the number of elements.
3. Read the elements of the array.

   ```
   scanf("%d",&n);
   for (int i=0;i<n;i++){
       scanf("%d",&arr[i]);
   }
   ```

4. Perform Mergesort Operation.
5. Print the obtained result.
6. Stop.

5.4 Output



5.5 Time Complexity Analysis

6 Prim's and Krushkal's Algorithm

6.1 Sample I/O

6.2 Analysis

6.3 Algorithm

6.4 Output

6.5 Time Complexity Analysis

7 Dijkstra's Algorithm

7.1 Sample I/O

7.2 Analysis

7.3 Algorithm

7.4 Output

7.5 Time Complexity Analysis

8 Matrix Chain Multiplication

8.1 Sample I/O

8.2 Analysis

8.3 Algorithm

8.4 Output

8.5 Time Complexity Analysis

9 0-1 Knapsack Problem

9.1 Sample I/O

9.2 Analysis

9.3 Algorithm

9.4 Output

9.5 Time Complexity Analysis

10 Travelling Salesman Problem

10.1 Sample I/O

10.2 Analysis

10.3 Algorithm

10.4 Output

10.5 Time Complexity Analysis

11 Longest Common Subsequence

11.1 Sample I/O

11.2 Analysis

11.3 Algorithm

11.4 Output

11.5 Time Complexity Analysis