Design and Analysis of Algorithms

Lab1(16-07-2021)

Assignment - Sorting

- R.Abhinav
- CB.EN.U4CSE19453

1. An array of n elements contains all but one of the integers from 1 to n + 1.

    1. Give the best algorithm you can for determining which number is missing if the array is sorted, and analyze its asymptotic worst-case running time.

Code:

```
#include <iostream>
using namespace std;

int check(int array[], int n)
{
    int i = 0, j = n - 1;
    int middle;
    while ((j - i) > 1)
    {
        middle = (i + j) / 2;
        if ((array[i] - i) != (array[middle] - middle))
            j = middle;
        else if ((array[j] - j) != (array[middle] - middle))
            i = middle;
    }
    return (array[i] + 1);
}

int main()
{
    int num;
    cout<<"Enter size of the array :"<<endl;
    cin >> num;
    cout<<"Enter the values:"<<endl;
    int arr[num];
    for (int i = 0; i < num; i++)
    {
        cin >> arr[i];
    }
    cout << "Missing number is:" << check(arr, num);
}
```

Output:

```
C:\Users\Administrator\Desktop\Untitled-1.exe
Enter size of the array :
5
Enter the values:
3
4
5
7
8
Missing number is:6
--------------------------------
Process exited after 15 seconds with return value 0
Press any key to continue . . .
```

Explaination:

1)

(1) Sorted Array :-

Number = (index + 1) ?

if True;

goto right subarray

false;

goto left subarray

Clearly;

if array is sorted, binary search can be used.

Check for smallest index (i) where $A[i] = i+1$

↓

this is our missing number.

if $A[\frac{n}{2}] = \frac{n}{2} + 1$

$i \leq \frac{n}{2}$; recurse 1st half of A

$i > \frac{n}{2}$; recurse 2nd half of A

from this we observe; $T(n) = T(\frac{n}{2}) + O(1)$,

hence; the worst case run-time = $\boxed{O(\log n)}$

2. Give the best algorithm you can for determining which number is missing if the array is not sorted, and analyze its asymptotic worst-case running time.

Code:

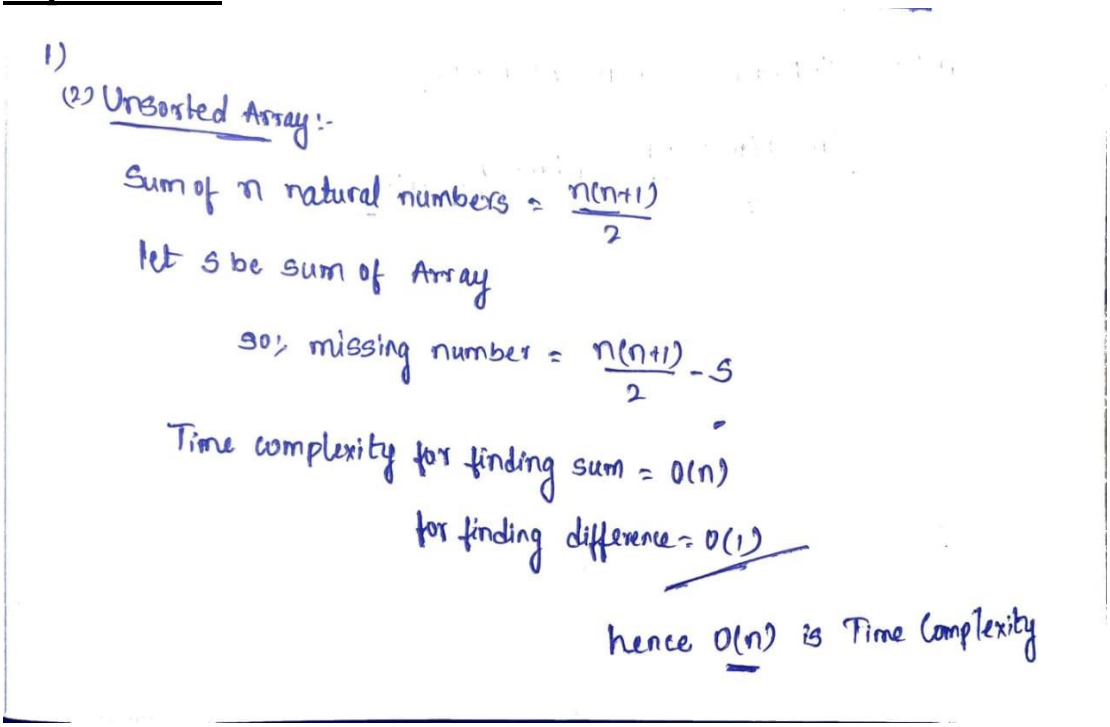
```
#include <iostream>
using namespace std;
int missed(int array[], int n)
{
    int a;
    a = (n + 1) * (n + 2) / 2;
    for (int i = 0; i < n; i++)
        a -= array[i];
    return a;
}
int main()
{
    int num;
    cout << "Enter size of the array :" << endl;
    cin >> num;
    int arr[num];
    cout << "Enter the values:" << endl;
    for (int i = 0; i < num; i++)
    {
        cin >> arr[i];
    }
    cout << "\n"
         << endl;
    int m = missed(arr, num);
    cout << "Missing number is: " << m;
}
```

Output:

```
C:\Users\Administrator\Desktop\Untitled-1.exe
Enter size of the array :
5
Enter the values:
2
4
1
5
6


Missing number is: 3
--------------------------------
Process exited after 9.997 seconds with return value 0
Press any key to continue . . .
```

Explaination:



2. Insertion Sort :

Code:

```c
#include <stdio.h>
int binary(int a[], int e, int h, int l)
{
    if (h <= l)
        return (e > a[l]) ? (l + 1) : l;

    int middleelement = (l + h) / 2;

    if (e == a[middleelement])
        return middleelement + 1;

    if (e > a[middleelement])
        return binary(a, e, middleelement + 1, h);
    return binary(a, e, l, middleelement - 1);
}
void insertion(int a[], int n)
{
    int i, b, j, k, s;

    for (i = 1; i < n; ++i)
    {
        j = i - 1;
        s = a[i];
        b = binary(a, s, 0, j);
        while (j >= b)
```

```
        {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = s;
    }
}

int main()
{
    int n, i;
    printf("enter size \n");
    scanf("%d", &n);
    int a[n];
    for (int i = 0; i < n; i++)
    {
        scanf("%d ", &a[i]);
    }

    insertion(a, n);

    printf("Sorted array: ");
    for (i = 0; i < n; i++)
    {
        printf("%d,", a[i]);
    }
    return 0;
}
```

Output:

```
C:\Users\Administrator\Desktop\q2.exe
Enter size of the array:
4
Enter elements:
3
1
5
7
9
Sorted array: 1,3,5,7,
--------------------------------
Process exited after 10.62 seconds with return value 0
Press any key to continue . . .
```

Explaination:

2) Insertion Sort :-

This sort runs with ; $O(n)$ time in best case

$O(n^2)$ time in avg & worst case.

(i) best case :-

Operations :- 1. It scans through the list

2. Comparing each pair of elements and it swaps (if not in order).

So; if array sorted no swaps ; hence $O(n)$

(ii) Worst case :-

The case comes when the list is in decreasing order

last element insertion;

$(n-1)$ comparisons & swaps.

2nd last;

$(n-2)$ comparisons & swaps.

Total = $2 \times (1+2+\ldots+n-2+n-1)$

$= \frac{2(n-1)(n-1+1)}{2} = n(n-1) \Rightarrow O(n^2)$

Improvement:

Improvement :-

Can be improved by using binary search for searching the correct position of the element.

As explained above;

it takes O(n) comparisons at n^{th} iteration

here;

if we use binary search :

the worst case scenario can be reduced to $\theta(\log n)$