DAA AS#2

0801CS191043 - Kunal Anand

Quick Sorting.

1 - Taking first element as pivot

```
// ----- first -----

int comp =0;
int partition_first(int Ar[],int start, int end)
{
   int pivot = Ar[start];
   int i = start;
   int j = end;


   while (i < j)
   {
      while (Ar[i] <= pivot) i++,comp++;
      while (Ar[j] > pivot) j--,comp++;


      if (i < j)
         swap(Ar[i], Ar[j]);
   }
   swap(Ar[start], Ar[j]);
   return j;
}
void quickSort_first(int Ar[],int start, int end)
{
   if (start < end)
   {
      int pivot = partition_first(Ar,start, end);


      quickSort_first(Ar,start, pivot - 1);
      quickSort_first(Ar,pivot + 1, end);
   }
}
```

Analysis -

Let’s take the worst case (partition is at first or at the end)
So for the next step we will have one less element only.
So, T(n) = T(n-1) + constant(n)

Worst case will appear when we try to sort the already sorted array (asc or desc) taking first or last element as pivot.

So the recursion tree will go like n -> (n-1) -> (n-2) -> (n-3) ..... -> 1
The sum will be n(n+1)/2

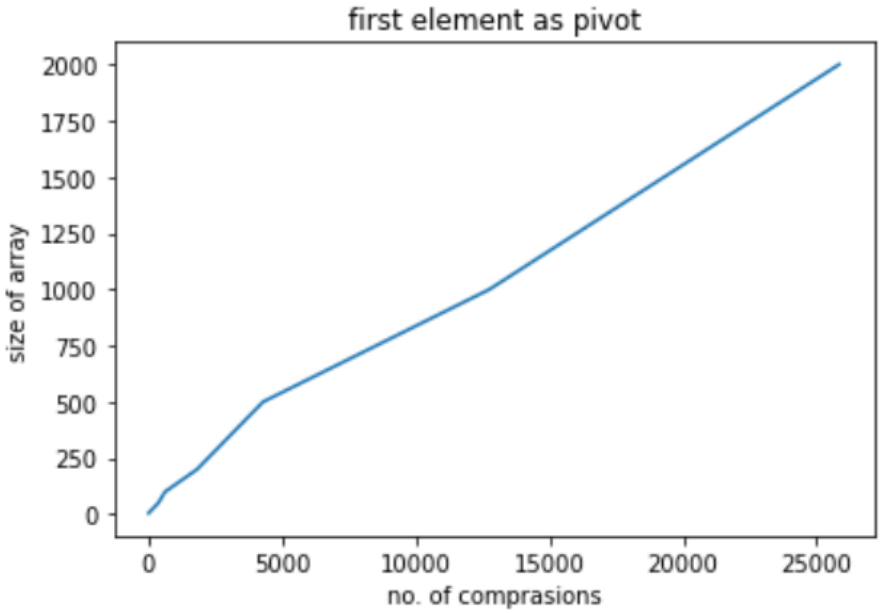
Which gives us the time complexity as O(n^2)

In the best case we will have to partition the array at the middle on a sorted array so that the resultant place of the pivot element will come at its place(middle).
So in this case we are dividing it into two parts in each step till it becomes 1 so height will be log(n).
Same as merge sorting.

T(n) = 2T(n/2) + constant(n)

Which gives us the time complexity as O(nlogn)

size of array	number of comparison
5	8
50	369
100	632
200	1822
500	4290
1000	12756
2000	25834

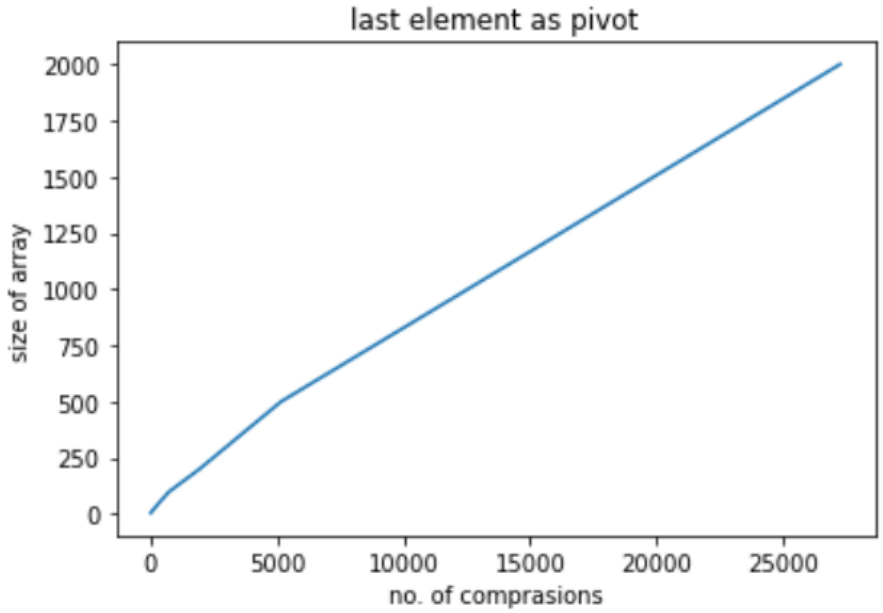


2 - Taking last element as pivot

```
// ----- last -----

int partition_last(int Ar[],int start, int end)
{
    int pivot = Ar[end];
    int i = start;
    int j = end;
    while (i < j)
    {
        while (Ar[i] < pivot) i++,comp++;
        while (Ar[j] >= pivot) j--,comp++;


        if (i < j)
            swap(Ar[i], Ar[j]);
    }
    swap(Ar[end], Ar[i]);
    return i;
}
void quickSort_last(int Ar[],int start, int end)
```

```
{
    if (start < end)
    {
        int pivot_element_index = partition_last(Ar,start, end);


        quickSort_last(Ar,start, pivot_element_index - 1);
        quickSort_last(Ar,pivot_element_index + 1, end);
    }
}
```

size of array	number of comparison
5	9
50	341
100	744
200	1936
500	5160
1000	12738
2000	27249

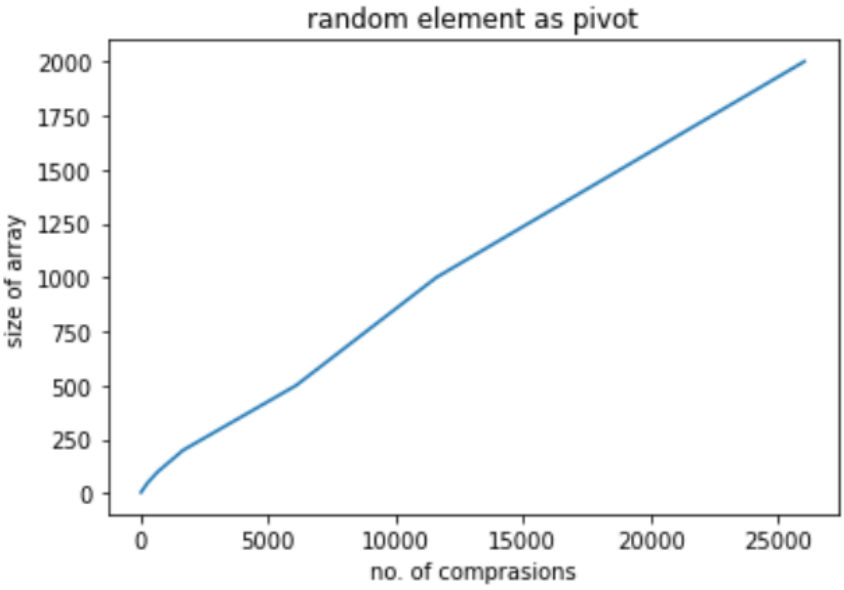


3 - Taking random element as pivot

```
int partition_random(int Ar[],int start, int end)
{
    int pivot = Ar[end];
    int i = start;
    int j = end;
    while (i < j)
    {
        while (Ar[i] < pivot) i++,comp++;
        while (Ar[j] >= pivot) j--,comp++;


        if (i < j)
            swap(Ar[i], Ar[j]);
    }
    swap(Ar[end], Ar[i]);
    return i;
}


int partition_r(int arr[], int low, int high)
{
    srand(time(NULL));
    int random = low + rand() % (high - low);


    swap(arr[random], arr[high]);


    return partition_random(arr, low, high);
}


void quickSort_random(int arr[], int low, int high)
{
```

```
    if (low < high) {

        int pi = partition_r(arr, low, high);


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

size of array	number of comparison
5	14
50	282
100	668
200	1664
500	6094
1000	11594
2000	26020

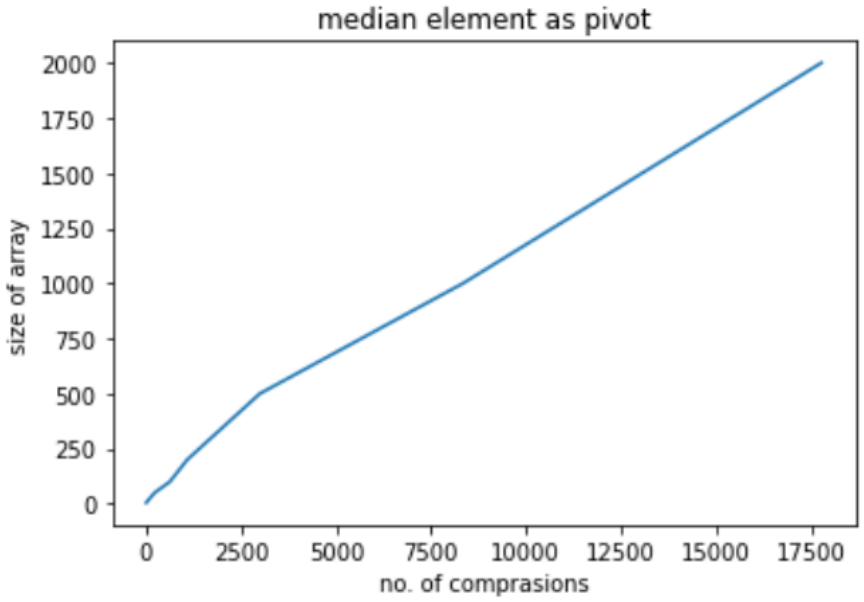


4 - Taking middle element as pivot

```
// ----- middle -----


void quickSort_middle(int arr[], int left, int right) {


    int i = left, j = right;
    int tmp;
    int pivot = arr[(left + right) / 2];


    while (i <= j) {
        while (arr[i] < pivot) i++,comp++;
        while (arr[j] > pivot) j--,comp++;


        if (i <= j) {
            tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;

            i++; j--;
```

```
            }
        };

    if (left < j)
      quickSort_middle(arr, left, j);

    if (i < right)
      quickSort_middle(arr, i, right);

}
```

size of array	number of comparison
5	5
50	225
100	627
200	1083
500	2996
1000	8228
2000	17764


median element as pivot
size of array
no. of comprasions
0
250
500
750
1000
1250
1500
1750
2000
0
2500
5000
7500
10000
12500
15000
17500