CS217 DSAA Homework-5

HONGLI YE 12311501

Contents

1 Question 1

Illustrate the operation of QuickSort on the array.

4	3	8	2	7	5	1	6

Table 1: Question 1

Write down the arguments for each recursive call to QuickSort (e. g. "QuickSort(A, 2, 5)") and the contents of the relevant subarray in each step of Partition (see Figure 7.1). Use vertical bars as in Figure 7.1 to indicate regions of values " $\leq x$" and " $> x$". You may leave out elements outside the relevant subarray and calls to QuickSort on subarrays of size 0 or 1.

Answer:

The Recursive Call is $QuickSort(A, 0, 7)$

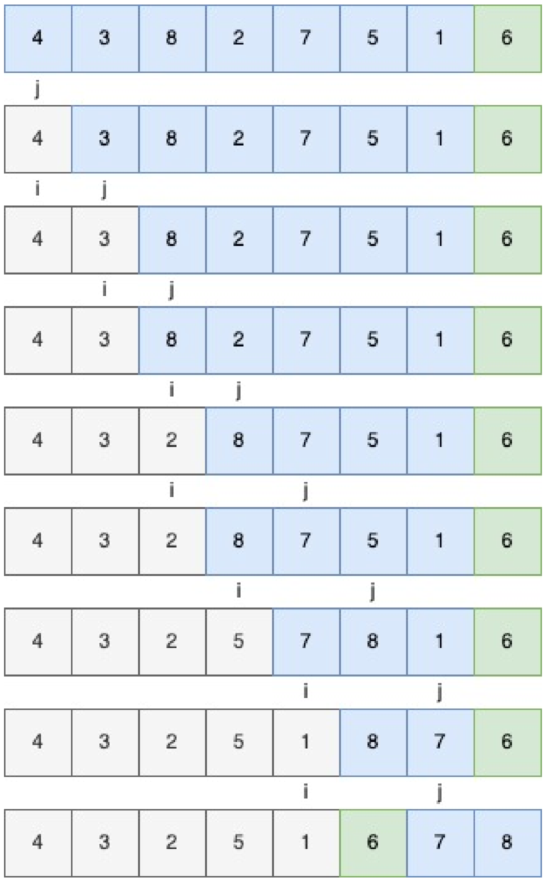

Figure 1: Call 1

The Recursive Calls are $QuickSort(A, 0, 4)$ and $QuickSort(A, 6, 7)$

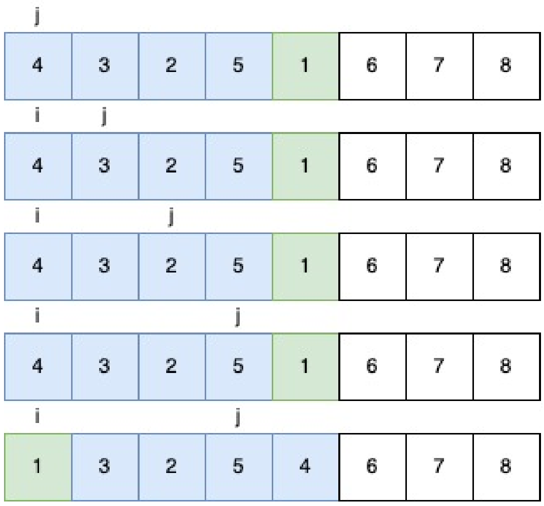

Figure 2: Call 2

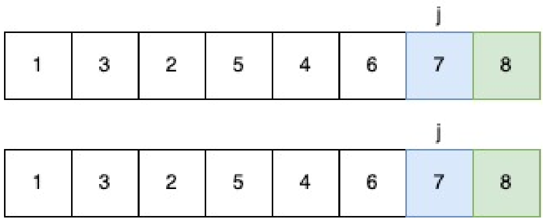

Figure 3: Call 3

The Recursive Call is $QuickSort(A, 1, 4)$

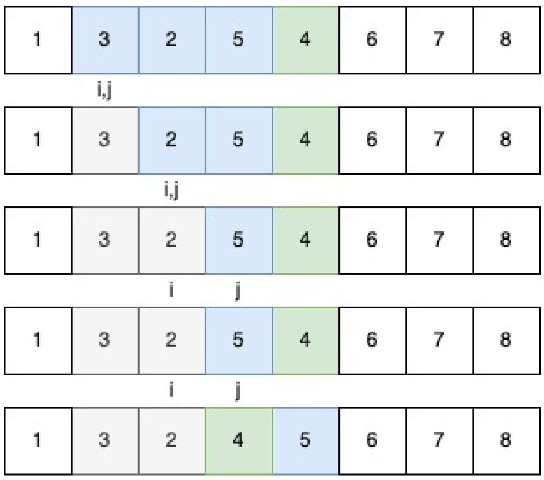

Figure 4: Call 4

The Recursive Call is $QuickSort(A, 1, 2)$

Figure 5: Call 5

2 Question 2

Prove that deterministic $QuickSort(A, p, r)$ is correct (you can use that $Partition$ is correct since that was proved at lecture)

Answer:

The partition function aims to sort the array into two parts where left side's maximum is less than the right side's minimum.

1. **Basic Statement:**
   The trivial situation of an array containing only one element is obviously sorted.

2. **Induction:**
   Suppose after i^{th} iterations, $A = [A_1, A_2, \ldots, A_k]$, where A_j are j^{th} subarray, and:
   $$\forall\ j_1 < j_2, max(A_{j_1}) < min(A_{j_2})$$
   Then we do $Partition$ for all A_j, we have a new $A = [A'_1, A'_2, \ldots, A'_{k'}]$, and by the property of Partition, $\forall\ j_1 < j_2, max(A'_{j_1}) < min(A'_{j_2})$ still holds.

3. **Termination:**
   Each time, the length of each array is strictly decreasing and the length of A is finite. So there must exist a finite number n, such that after n^{th} iteration, all subarray's length equal to 1. Which appeals to the basic statement.

3 Question 3

What is the runtime of QuickSort when the array A contains distinct elements sorted in decreasing order? (Justify your answer)

Answer:

Let $A = [a_1, a_2 \ldots, a_n]$ and we know that $a_i > a_j, \forall\ i < j$.
Let A_i be the new array after i^{th} iteration.
Let $T(k)$ be the runtime of $QuickSort(A, n-k, n-1)$
Let $P(k)$ be the runtime of $Partition(A, n-k, n-1)$

- **Begin:**
  The initial call is $QuickSort(A, 0, n-1)$. After the first iteration, $A_1 = [a_n, a_1, a_2 \ldots a_{n-1}]$. Then the second call is $QuickSort(A, 1, n-1)$, since the left subarray is a single element.

- **Induction:**
  Suppose the i^{th} call is $QuickSort(A, i-1, n-1)$. After the i^{th} iteration, $A_i = [a_n, \ldots, a_{n-i+1}, a_1, \ldots, a_{n-i}]$. Then the $i+1^{th}$ call is $QuickSort(A, i, n-1)$, since the left subarray is a single element.

- **Termination:**
  The last call is $QuickSort(A, n-1, n-1)$. After this call. $A_n = [a_n, a_{n-1}, \ldots, a_1]$ is sorted. So the induction is correct.

By the induction, we have:
$$T(n) = T(n-1) + P(n)$$
Use the knowledge of basic number array:
$$T(n) = \Sigma_{k=1}^{n} P(k)$$
Now we have a look at the code of $Partition$:

```
int Partition(int arr[], int p, int r)
```

```
{
    int middle = arr[r];
    int i = p-1;
    for(int j=p ; j <= r - 1 ; j++)
    {
        if(arr[j] <= middle)
        {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i+1], arr[r]);
    return i+1;
}
```

By the structure of the decreasing array. i doesn't change in the whole loop.

$$P(k) = c_1 + c_2 + ((n-1) - (n-k) + 1)c_3 + c_4 = C + c_3 k$$

So:

$$T(n) = c_3 \frac{(k+1)k}{2} + Ck$$

So:

$$T(n) = \Theta(n^2)$$

4 Question 4

What value of q does Partition return when all n elements have the same value? What is the asymptotic runtime (Θ-notation) of QuickSort for such an input? (Justify your answer)

Answer:

Assume $A = [a_1, a_2, \dots, a_n]$, where $a_i = p \ \forall i \in [1, n]$

```
int Partition(int arr[], int p, int r)
{
    int middle = arr[r];
    int i = p-1;
    for(int j=p ; j <= r - 1 ; j++)
    {
        if(arr[j] <= middle)
        {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i+1], arr[r]);
    return i+1;
}
```

Since the partition takes all the elements small or equal to the $arr[r]$ to the left. Then the runtime of the *QuickSort* is the same as **Question 4**. So: $T(n) = \Theta(n^2)$

5 Question 5

Modify Partition so it divides the subarray in three parts from left to right:

- $A[p \dots k]$ contains elements smaller than x
- $A[i+1 \dots k]$ contains elements equal to x.
- $A[k+1 \dots n]$ contains elements bigger than x.

Use pseudocode or your favourite programming language to write down your modified procedure Partition' and explain the idea(s) behind it. It should still run in $\Theta(n)$ time for every n-element subarray. Give a brief argument as to why that is the case. Partition' should return two variables q, t such that $A[q \dots t]$ contains all elements with the same value as the pivot (including the pivot itself).

Also, write down a modified algorithm QuickSort' that uses Partition' and q, t in such a way that it recurses only on strictly smaller and strictly larger elements.

What is the asymptotic runtime of QuickSort' on the input from Question 5.4?

Answer:

The **Partition'** in cpp:

```cpp
vector<int> Partition(int arr[], int p, int r)
{
    int middle = arr[r];
    int i = p-1;
    int l = r;
    for(int j = p ; j <= l-1 ; j++)
    {
        if(arr[j] < middle)
        {
            i++;
            swap(arr[i], arr[j]);
        }
        else if(arr[j] == middle)
        {
            l--;
            swap(arr[l], arr[j]);
            j--;
        }
    }

    for(int j=1 ; j <= r-l+1 ; j++)
    {
        swap(arr[i+j], arr[r-j+1]);
    }

    return {i+1, i+1+r-l};
}
```

The **QuickSort'** in cpp:

```cpp
void QuickSort(int arr[], int p, int r)
{
    if (p < r)
    {
```

```
        vector<int> q = Partition(arr, p, r);
        QuickSort(arr, p, q[0] - 1);
        QuickSort(arr, q[1] + 1, r);
    }
}
```

The initial call is $QuickSort(arr, 0, n-1)$. Since all element are with same vaules, the second round calls are $QuickSort(arr, 0, 0)$ and $QuickSort(arr, n-1, n-1)$. So:

$$T(n) = \Theta(n)$$

6 Question 6

Answer: The code of Original QuickSort.

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

int Partition(int arr[], int p, int r)
{
    int middle = arr[r];
    int i = p-1;
    for(int j=p ; j <= r - 1 ; j++)
    {
        if(arr[j] <= middle)
        {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i+1], arr[r]);
    return i+1;
}
void QuickSort(int arr[], int p, int r)
{
    if (p < r)
    {
        int q = Partition(arr, p, r);
        QuickSort(arr, p, q - 1);
        QuickSort(arr, q + 1, r);
    }
}
int main()
{
    int n;
    cin>>n;
    int arr[n];
    for(int i=0;i<n;i++)
    {
        cin>>arr[i];
    }
```

```

    QuickSort(arr, 0, n-1);

    for(int i=0;i<n;i++)
    {
        cout<<arr[i]<<" ";
    }
    return 0;
}
```