Assignment V - DSAA(H)

Name: Yuxuan HOU (侯宇轩)

Student ID: 12413104

Date: 2025.10.17

Question 5.1 (Marks: 0.25)

Illustrate the operation of QUICKSORT on the array

4	3	8	2	7	5	1	6

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.

Sol:

QuickSort (A, 1, 8)

```
|4 3 8 2 7 5 1| 6
 4| 3 8 2 7 5 1 6
 4 3| 8 2 7 5 1 6
 4 3| 8| 2 7 5 1 6
 4 3 2| 8| 7 5 1 6
 4 3 2| 8 7| 5 1 6
 4 3 2 5| 7 8| 1 6
 4 3 2 5 1| 8 7| 6
 4 3 2 5 1| 6| 7 8
```

QuickSort (A, 1, 5)

```
|4 3 2 5| 1
|4| 3 2 5| 1
|4 3| 2 5| 1
|4 3 2| 5| 1
|1| 3 2 5 4
```

QuickSort (A, 2, 5)

```
|3 2 5| 4
 3| 2 5| 4
 3 2| 5| 4
 3 2| 5| 4
 3 2| 4| 5
```

QuickSort (A, 2, 3)

|3|2

|3|2

|2|3

QuickSort (A, 7, 8)

|7|8

7|8

7|8|

Question 5.2 (Marks: 0.5)

Prove that deterministic QUICKSORT(A, p, r) is correct (you can use that PARTITION is correct since that was proved at lecture).

```
QUICKSORT(A, p, r)
if p < r then
        q = PARTITION(A, p, r)
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q + 1, r)
```

PF:

Recursive Invariant: In each recursion, after `Partition(A, p, r)`, $A[p, q-1] \leq A[q] \leq A[q+1, r]$, and the following recursions only affect the two sides.

Initialization: At the beginning of recursion, the function will directly return if $p \geq r$, which is trivially correct. Otherwise satisfies the invariant due to the correctness of `Partition(A, p, r)`.

Maintenance: The following two recurisons make $A[p, q-1]$ and $A[q+1, r]$ ordered, and `Partition(A, p, r)` make sure $A[p, q-1] \leq A[q] \leq A[q+1, r]$, thus the array $A[p, r]$ will be ordered.

Termination: $r - p$ will trivially decrease, and when $p \geq r$, the recursion will terminate.

`Q.E.D.`.

Question 5.3 (Marks: 0.25)

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

Sol:

Obviously the case in description is the worst case, whose runtime is $\Theta(n^2)$.

Justification: Each partition will pick the minimum element as pivot, leaving the worst partition $(1, n-1)$, and the Partition itself is linear.

Thus we have: $T(n) = T(n-1) + \Theta(n)$, then solve it by substitution, we obtain $T(n) = \Theta(n^2)$.

Question 5.4 (Marks: 0.5)

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).

Sol: $q = r$, for each elements satisfy $A[j] \leq x$, thus $i \leftarrow i + 1$ will always be executed, thus Partition will return r.

The runtime will be $\Theta(n^2)$, similiar to Question 5.3, this situation will leave the worst partition $(1, n-1)$, and the Partition itself is linear. Thus we have: $T(n) = T(n-1) + \Theta(n)$, then solve it by substitution, we obtain $T(n) = \Theta(n^2)$.

Question 5.5 (Marks: 0.5)

Question 5.5 (Marks: 0.5)

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

- $A[p \dots i]$ contains elements smaller than x
- $A[i+1 \dots k]$ contains elements equal to x and
- $A[k+1 \dots j-1]$ contains elements larger 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?

Sol:

Code in C++:

```cpp
auto Partition = [](vector < int > &A, int l, int r)->pair < int, int >{
    int pivot(A[r]);
    int spl1(l - 1), spl2(r - 1);
    int cur(l);
    while(cur <= spl2){
        if(A[cur] < pivot)swap(A[++spl1], A[cur++]);
        else if(A[cur] > pivot)swap(A[cur], A[spl2--]);
        else ++cur;
    }
    swap(A[++spl2], A[r]);
    return {spl1 + 1, spl2};
};
```

Explaination: `cur` will traverse each elements in $A[l, r-1]$, if it's less than pivot, then $spl1 \leftarrow spl1 + 1$, and we implement the swap, i.e., the element will be placed in the less range. Simutaneously, if it's larger, it will be replaced to the last $spl2$, which is the edge of the larger range. And for equal elements will be left between the two spl. Finally, swap the pivot to the rightest of the middle range. Therefore, we have $A[l, spl1] < A[spl1+1, spl2] < A[spl2+1, r]$, which satisfy the description, and the runtime is trivially linear, for $spl2 - cur$ will definitely decrease in each while loop dur to the `cur++`, `spl2--` and `++cur`, thus the algorithm must terminate in linear runtime.

Code in C++:

```cpp
auto QuickSort = [&](auto&& self, vector < int > &A, int l, int r)->void{
    if(l >= r)return;
    auto [spl1, spl2] = Partition(A, l, r);
    self(self, A, l, spl1 - 1);
    self(self, A, spl2 + 1, r);
};
```

The runtime will be $\Theta(n)$, for Partition' will return `{l, r}` because all the elements are equal to the pivot, then the QuickSort' will not enter the recursion. Thus the runtime will only be once of the Partition'. which is $\Theta(n)$.

Question 5.6 (Marks: 0.5)

Implement QuickSort and QuickSort$'$ from Question 5.5.

题目

状态	最后递交于	题目
✓ 100 Accepted	2 天前	**31** Quick Sort I
✓ 100 Accepted	16 小时前	**32** Quick Sort II

```cpp
int main(){
```

```
    int N = read();
    vector < int > A(N + 10, 0);
    for(int i = 1; i <= N; ++i)A[i] = read();
    auto Partition = [](vector < int > &A, int l, int r)->int{
        int val(A[r]);
        int spl(l - 1);
        for(int i = l; i <= r - 1; ++i)
            if(A[i] <= val)swap(A[++spl], A[i]);
        swap(A[++spl], A[r]);
        return spl;
    };
    auto QuickSort = [&](auto&& self, vector < int > &A, int l, int r)->void{
        if(l >= r)return;
        int spl = Partition(A, l, r);
        self(self, A, l, spl - 1);
        self(self, A, spl + 1, r);
    }; QuickSort(QuickSort, A, 1, N);

    for(int i = 1; i <= N; ++i)printf("%d%c", A[i], i == N ? '\n' : ' ');

    // fprintf(stderr, "Time: %.6lf\n", (double)clock() / CLOCKS_PER_SEC);
    return 0;
}
```

```
int main(){
    int N = read();
    vector < int > A(N + 10, 0);
    for(int i = 1; i <= N; ++i)A[i] = read();
    auto Partition = [](vector < int > &A, int l, int r)->pair < int, int >{
        int pivot(A[r]);
        int spl1(l - 1), spl2(r - 1);
        int cur(l);
        while(cur <= spl2){
            if(A[cur] < pivot)swap(A[++spl1], A[cur++]);
            else if(A[cur] > pivot)swap(A[cur], A[spl2--]);
```

```
            else ++cur;
        }
        swap(A[++spl2], A[r]);
        return {spl1 + 1, spl2};
    };
    auto QuickSort = [&](auto&& self, vector < int > &A, int l, int r)->void{
        if(l >= r)return;
        auto [spl1, spl2] = Partition(A, l, r);
        self(self, A, l, spl1 - 1);
        self(self, A, spl2 + 1, r);
    }; QuickSort(QuickSort, A, 1, N);

    for(int i = 1; i <= N; ++i)printf("%d%c", A[i], i == N ? '\n' : ' ');

    // fprintf(stderr, "Time: %.6lf\n", (double)clock() / CLOCKS_PER_SEC);
    return 0;
}
```