排序演算法

合併排序、快速排序……

nevikw39

點石學園

August 25, 2021

- 1 序
 - Bubble Sort
- 2 原理
 - Merge Sort
 - Inversion
 - Quicksort
 - Quickselect
- 3 實務
 - std::sort()

Section 1

序

- 1 序
 - Bubble Sort
- 2 原理
 - Merge Sort
 - Quicksort
- 3 實務

Introduction to Sorting

Definition (Sort)

Rearrange elements in an array into a sort of order.

- Monotonicity
- Permutation of original array

Introduction to Sorting

Definition (Sort)

Rearrange elements in an array into a sort of order.

- Monotonicity
- Permutation of original array

Reason

- Ranking
- \bullet Prerequisite of other algorithms such as binary search, greedy, \dots

Naïve Approach: Bubble Sort

```
procedure Bubble Sort(\{a_0, a_1, ..., a_{n-1}\})

for i \in [0, n-1) do

for j \in [0, n-1-i) do

if a_j > a_{j+1} then

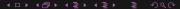
 \text{SWAP}(a_j, a_{j+1})
end if
end for
end for
return a
end procedure
```



5/33

nevikw39 (點石學園)

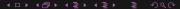
8 27 2021 110 20



8 27 2021 110 20



8 27 2021 110 20



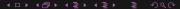
8 27 2021 110 20



8 27 110 2021 20

8 27 110 2021 20

8 27 110 20 2021



8 27 110 20 2021

8 27 110 20 2021



8 27 110 20 202



8 27 20 110 2021



8 27 20 110 202

8 27 20 110 202



8 20 27 110 2021



8 20 27 110 2021



Section 2

原理

- 1 序
 - Bubble Sort
- 2 原理
 - Merge Sort
 - Inversion
 - Quicksort
 - Quickselect
- 3 實務



分治三部曲:

分治三部曲:

Divide Split original problem into several subproblems

分治三部曲:

Divide Split original problem into several subproblems

Conquer Recur to each subproblem until it could be easily solved

分治三部曲:

Divide Split original problem into several subproblems

Conquer Recur to each subproblem until it could be easily solved

Combine Merge the results of subproblems

分治三部曲:

Divide Split original problem into several subproblems

Conquer Recur to each subproblem until it could be easily solved

Combine Merge the results of subproblems

時間複雜度: Master Theorem

分治三部曲:

Divide Split original problem into several subproblems

Conquer Recur to each subproblem until it could be easily solved

Combine Merge the results of subproblems

時間複雜度: Master Theorem

$$T(n) = aT(\frac{n}{b}) + O(n^d) = \begin{cases} O(n^d), & d > \log_b a \\ O(n^d \log n), & d = \log_b a \\ O(n^{\log_b a}), & d < \log_b a \end{cases}$$

合併排序

Divide Split array into left and right subarrays (b = 2, O(1))

合併排序

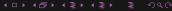
Divide Split array into left and right subarrays (b = 2, O(1)) Conquer Sort two subarrays recursively (a = 2)

◆ロト ◆個ト ◆夏 ト ◆夏 ト 夏 めのの

Divide Split array into left and right subarrays (b=2,O(1))

Conquer Sort two subarrays recursively (a = 2)

Combine Merge two sorted subarrays in O(n)



nevikw39 (點石學園)

Given two sorted subarrays *left*, *right*, how could we combine them to a sorted array efficiently??

Given two sorted subarrays left, right, how could we combine them to a sorted array efficiently??

Let **itr**, **jtr** point to the begin of *left*, *right* respectively.

nevikw39 (點石學園)

Given two sorted subarrays *left*, *right*, how could we combine them to a sorted array efficiently??

Let itr, jtr point to the begin of *left*, *right* respectively.

If $itr \neq the$ end of $left \land jtr \neq the$ end of right, then we choose the min one and forward the pointer.

nevikw39 (點石學園)

Given two sorted subarrays *left*, *right*, how could we combine them to a sorted array efficiently??

Let **itr**, **jtr** point to the begin of *left*, *right* respectively.

If $\mathbf{itr} \neq \text{the end of } left \land \mathbf{jtr} \neq \text{the end of } right$, then we choose the min one and forward the pointer.

Else if $\mathbf{itr} \neq \text{the end of } left \lor \mathbf{jtr} \neq \text{the end of } right$, then we choose the pointer and forward it.

Given two sorted subarrays *left*, *right*, how could we combine them to a sorted array efficiently??

Let **itr**, **jtr** point to the begin of *left*, *right* respectively.

If $itr \neq the$ end of $left \land jtr \neq the$ end of right, then we choose the min one and forward the pointer.

Else if $\mathbf{itr} \neq \text{the end of } left \lor \mathbf{jtr} \neq \text{the end of } right$, then we choose the pointer and forward it.

Repeat until itr = the end of $left \land jtr$ = the end of right.

procedure MERGE SORT(*begin, *end})

```
procedure MERGE SORT(*begin, *end})
   if end - begin = 1 \text{ then}
      return
   end if
```

▷ 0. Recursion boundary

```
procedure MERGE SORT(*begin, *end})

if end - begin = 1 then \triangleright 0. Recursion boundary

return

end if

mid \leftarrow \frac{begin + end}{2}, left \leftarrow [begin, mid), right \leftarrow [mid, end) \triangleright 1. Divide
```

```
procedure Merge Sort(*begin, *end})

if end - begin = 1 then \triangleright 0. Recursion boundary

return

end if

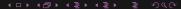
mid \leftarrow \frac{begin + end}{2}, left \leftarrow [begin, mid), right \leftarrow [mid, end) \triangleright 1. Divide

Merge Sort(left.begin(), left.end()) \triangleright 2. Conquer

Merge Sort(right.begin(), right.end())
```

```
procedure Merge Sort(*begin, *end})
    if end - begin = 1 \text{ then}
                                                           \triangleright 0. Recursion boundary
         return
    end if
    mid \leftarrow \frac{begin + end}{2}, left \leftarrow [begin, mid), right \leftarrow [mid, end) \triangleright 1. Divide
    MERGE SORT(left.begin(), left.end())
                                                                          \triangleright 2. Conquer
    MERGE SORT(right.begin(), right.end())
    itr \leftarrow left.begin(), jtr \leftarrow right.begin()
    while begin \neq end do
                                                                          \triangleright 3. Combine
         if itr \neq left.end() \land (jtr = right.end() \lor *itr < *jtr) then
             *begin ++ \leftarrow *itr ++
         else
             *begin ++ \leftarrow *itr ++
         end if
```

end while



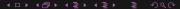
8 27

2021

110 20

8 27 2021 110 20





類題演練

- AtCoder Beginner Contest 154 p. C Distinct or Not
- LeetCode 0004. Median of Two Sorted Arrays (Hard!?)

4□▶ 4□▶ 4 亘▶ 4 亘 ▶ 9

Distinct or Not

AtCoder Beginner Contest 154 p. C

Problem

Determine if each element in the array is unique.

Distinct or Not

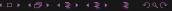
AtCoder Beginner Contest 154 p. C

Problem

Determine if each element in the array is unique.

Idea

Use nested loop??



Distinct or Not

AtCoder Beginner Contest 154 p. C

Problem

Determine if each element in the array is unique.

Idea

Use nested loop??

Algorithm

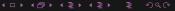
In **combine** process of Merge Sort, we could check whether *itr == *jtr.

Median of Two Sorted Arrays

LeetCode 0004 (Hard!?)

Definition (Median)

For an array a whose size is n, if n is odd, then its median is $a_{n/2}$; otherwise, it is $\frac{a_{n/2} + a_{n/2-1}}{2}$



nevikw39 (點石學園) 排序演算法 August 25, 2021 15/33

Median of Two Sorted Arrays

LeetCode 0004 (Hard!?)

Definition (Median)

For an array a whose size is n, if n is odd, then its median is $a_{n/2}$; otherwise, it is $\frac{a_{n/2}+a_{n/2-1}}{2}$

Algorithm

Obviously, all we have to do is to perform **combine** process of MERGE SORT.

□▶ 4屆▶ 4절▶ 4 절▶

Definition (Inversion)

Given a sequence S. If $i < j \iff S_i > S_j$, then (i, j) or (S_i, S_j) is called a **inversion** of S.

Definition (Inversion)

Given a sequence S. If $i < j \iff S_i > S_j$, then (i, j) or (S_i, S_j) is called a **inversion** of S.

Idea

 $\forall i \in [0, |S|), \forall j \in [i, |S|), \text{ count whether } i < j \iff S_i > S_j??$

Definition (Inversion)

Given a sequence S. If $i < j \iff S_i > S_j$, then (i, j) or (S_i, S_j) is called a **inversion** of S.

Idea

$$\forall i \in [0, |S|), \, \forall j \in [i, |S|), \, \text{count whether} \,\, i < j \iff S_i > S_j ??$$

Algorithm

When **combining** two sorted subarrays, if *itr > *jtr, then a inversion exists, and *(itr + 1), ... are also greater than *jtr with a smaller index.

類題演練

- AtCoder Beginner Contest 190 p. F Shift and Inversions
- UVa 10810: Ultra-QuickSort
- UVa 11858: Frosh Week

AtCoder Beginner Contest 190 p. F

 $v = \{0, 1, 2, 3, 4\}$, there are 0 inversions.

AtCoder Beginner Contest 190 p. F

- $v = \{0, 1, 2, 3, 4\}$, there are 0 inversions.
- $v = \{1, 2, 3, 4, 0\}, \text{ there are 4 inversions: } (1, 0), (2, 0), (3, 0), (4, 0)$

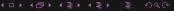
- $v = \{0, 1, 2, 3, 4\}$, there are 0 inversions.
- $v = \{1, 2, 3, 4, 0\}, \text{ there are 4 inversions: } (1, 0), (2, 0), (3, 0), (4, 0)$
- $v = \{2, 3, 4, 0, 1\}$, there are 6 inversions: (2, 0), (3, 0), (4, 0), (2, 1), (3, 1), (4, 1)

- $v = \{0, 1, 2, 3, 4\}$, there are 0 inversions.
- $v = \{1, 2, 3, 4, 0\}, \text{ there are 4 inversions: } (1, 0), (2, 0), (3, 0), (4, 0)$
- $v = \{2, 3, 4, 0, 1\}$, there are 6 inversions: (2,0),(3,0),(4,0),(2,1),(3,1),(4,1)
- $v = \{3, 4, 0, 1, 2\}$, there are 6 inversions: (3,0),(3,1),(4,0),(4,1),(3,2),(4,2)

- $v = \{0, 1, 2, 3, 4\}$, there are 0 inversions.
- $v = \{1, 2, 3, 4, 0\}, \text{ there are 4 inversions: } (1, 0), (2, 0), (3, 0), (4, 0)$
- $v = \{2, 3, 4, 0, 1\}$, there are 6 inversions: (2,0),(3,0),(4,0),(2,1),(3,1),(4,1)
- $v = \{3, 4, 0, 1, 2\}$, there are 6 inversions: (3,0),(3,1),(4,0),(4,1),(3,2),(4,2)
- $v = \{4, 0, 1, 2, \overline{3}\}, \text{ there are 4 inversions: } (4, 0), (4, 1), (4, 2), (4, 3)$

AtCoder Beginner Contest 190 p. F

- $v = \{0, 1, 2, 3, 4\}$, there are 0 inversions.
- $v = \{1, 2, 3, 4, 0\}, \text{ there are 4 inversions: } (1, 0), (2, 0), (3, 0), (4, 0)$
- $v = \{2, 3, 4, 0, 1\}$, there are 6 inversions: (2, 0), (3, 0), (4, 0), (2, 1), (3, 1), (4, 1)
- $v = \{3, 4, 0, 1, 2\}, \text{ there are 6 inversions:}$ (3, 0), (3, 1), (4, 0), (4, 1), (3, 2), (4, 2)
- $v = \{4, 0, 1, 2, 3\}, \text{ there are 4 inversions: } (4, 0), (4, 1), (4, 2), (4, 3)$
- When removing a from the front of the array, the number of inversion would decrease by a for a is the a-th smallest element in the array.



18/33

- $v = \{0, 1, 2, 3, 4\}$, there are 0 inversions.
- $v = \{1, 2, 3, 4, 0\}, \text{ there are 4 inversions: } (1, 0), (2, 0), (3, 0), (4, 0)$
- $v = \{2, 3, 4, 0, 1\}$, there are 6 inversions: (2, 0), (3, 0), (4, 0), (2, 1), (3, 1), (4, 1)
- $v = \{3, 4, 0, 1, 2\}$, there are 6 inversions: (3, 0), (3, 1), (4, 0), (4, 1), (3, 2), (4, 2)
- $v = \{4, 0, 1, 2, 3\}, \text{ there are 4 inversions: } (4, 0), (4, 1), (4, 2), (4, 3)$
- When removing a from the front of the array, the number of inversion would decrease by a for a is the a-th smallest element in the array.
- In like manner, when adding a to the back of the array, the number would increase by N-1-a because a is less than N-1-a elements in the array.

快速排序

Divide Select a pivot value and separate the array into less partition and greater partition (b = 2, O(n))

快速排序

Divide Select a pivot value and separate the array into less partition and greater partition (b = 2, O(n))

Conquer Sort two partitions recursively (a = 2)

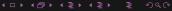


快速排序

Divide Select a *pivot* value and separate the array into *less* partition and *greater* partition (b = 2, O(n))

Conquer Sort two partitions recursively (a = 2)

Combine Nothing to do, the array have been sorted. (O(1))



原地分割 (In-place Division)

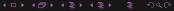
How could we separate the array into *less* partition and *greater* partition??



原地分割 (In-place Division)

How could we separate the array into less partition and greater partition??

Suppose two slices are [begin, itr) and [itr + 1, end).



How could we separate the array into *less* partition and *greater* partition??

Suppose two slices are [begin, itr) and [itr + 1, end).

Initially, $itr \leftarrow begin$, $pivot \leftarrow end - 1$.

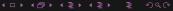


How could we separate the array into *less* partition and *greater* partition??

Suppose two slices are [begin, itr) and [itr+1, end).

Initially, $itr \leftarrow begin, \ pivot \leftarrow end - 1.$

We traverse from begin through pivot with jtr.



How could we separate the array into *less* partition and *greater* partition??

Suppose two slices are [begin, itr) and [itr + 1, end).

Initially, $itr \leftarrow begin, pivot \leftarrow end - 1$.

We traverse from begin through pivot with jtr.

If *jtr < *pivot, then we put it to *itr and forward itr.



How could we separate the array into *less* partition and *greater* partition??

Suppose two slices are [begin, itr) and [itr + 1, end).

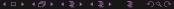
Initially, $itr \leftarrow begin, pivot \leftarrow end - 1$.

We traverse from begin through pivot with jtr.

If *jtr < *pivot, then we put it to *itr and forward itr.

Finally, swap *itr and *pivot.

The choice of *pivot* plays a significant role when it comes to the efficiency of QUICKSORT.



The choice of *pivot* plays a significant role when it comes to the efficiency of QUICKSORT.

In classical textbook "Introduction to Algorithm", C., S., L., R. chose the last element to be *pivot*.

The choice of pivot plays a significant role when it comes to the efficiency of Quicksort.

In classical textbook "Introduction to Algorithm", C., S., L., R. chose the last element to be pivot.

Nevertheless, fixed pivot may run into troubles because if unfortunately pivot is minimum or maximum every time, the time complexity would decline to $O(n^2)$.

The choice of *pivot* plays a significant role when it comes to the efficiency of QUICKSORT.

In classical textbook "Introduction to Algorithm", C., S., L., R. chose the last element to be *pivot*.

Nevertheless, fixed *pivot* may run into troubles because if unfortunately *pivot* is minimum or maximum every time, the time complexity would decline to $O(n^2)$.

With an eye to avoiding this, we can pick *pivot* randomly or use the median of $\{*begin, *(\frac{begin+end}{2}), *(end-1)\}$.

 $\mathbf{procedure} \ \mathrm{Quicksort}(\{*begin,*end\})$

```
procedure Quicksort(\{*begin, *end\})
if end - begin \le 1 then
return
end if
```

▷ 0. Recursion boundary

◆□▶ ◆□▶ ◆臺▶ ◆臺▶ 臺 か९@

```
procedure Quicksort(\{*begin, *end\})
   if end - begin < 1 then
       return
   end if
    itr \leftarrow begin, pivot \leftarrow end - 1
   for jtr \leftarrow begin; jtr \neq pivot; jtr ++ do
       if *jtr < *pivot then
           SWAP(*itr++,*jtr)
       end if
   end for
   SWAP(*itr, *pivot)
```

 \triangleright 0. Recursion boundary

▷ 1. Divide

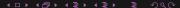
□ ▶ < □ ▶ < 亘 ▶ < 亘 ▶ ○ ○

```
procedure QUICKSORT(\{*begin, *end\})
   if end - begin < 1 then
                                                    \triangleright 0. Recursion boundary
       return
   end if
    itr \leftarrow begin, pivot \leftarrow end - 1
                                                                    ▶ 1. Divide
   for jtr \leftarrow begin; jtr \neq pivot; jtr +++ do
       if *jtr < *pivot then
           SWAP(*itr++,*jtr)
       end if
   end for
   SWAP(*itr, *pivot)
    Quicksort (begin, itr)
                                                                 \triangleright 2. Conquer
    Quicksort(itr + 1, end)
```

```
procedure QUICKSORT(\{*begin, *end\})
    if end - begin < 1 then
                                                      \triangleright 0. Recursion boundary
        return
    end if
    itr \leftarrow begin, pivot \leftarrow end - 1
                                                                       ▶ 1. Divide
    for jtr \leftarrow begin; jtr \neq pivot; jtr \leftrightarrow do
        if *jtr < *pivot then
            SWAP(*itr++,*jtr)
        end if
    end for
    SWAP(*itr, *pivot)
    Quicksort (begin, itr)
                                                                    \triangleright 2. Conquer
    Quicksort(itr + 1, end)
                                                                    \triangleright 3. Combine
end procedure
```

8 27 2021 110 20

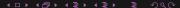
8 20 27 2021 110



 8
 20

 27
 110

 2021

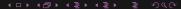


8

20

27 110 202





快速選擇 Quickselect

Problem

Find the k-th element in the array.

快速選擇 Quickselect

Problem

Find the k-th element in the array.

Idea

Sort the array??

快速選擇 Quickselect

Problem

Find the k-th element in the array.

Idea

Sort the array??

Algorithm

When **dividing** array into *less* and *greater* parts in Quicksort, we could know the size of them. If k < |less|, we only need to recur to less part and greater one could be ignored; vice versa.

Note that if we recur to greater part, we should update k to k - |less| - 1 because the k-th element in the original array is k - |less| - 1-th element in the greater part.

類題演練

- LeetCode 0215. Kth Largest Element in an Array (Medium)
- AtCoder Beginner Contest 161 p. C Popular Vote

◄□▶◀∰▶◀臺▶◀臺▶ 臺 ∽Q

Kth Largest Element in an Array

LeetCode 0215 (Medium)

非常單純的模板題,我們以此來示範 pivot 之選擇。

Always last 40ms, beats 17.10% submissions

Randomly 8ms, faster than 75.44% codes



Section 3

實務

- 1)序
 - Bubble Sort
- 2 原理
 - Merge Sort
 - Quicksort
- 3 實務
 - std::sort()



std::merge()

```
void merge_sort(vector<int>::iterator begin,
   vector < int > :: iterator end)
    // 0. recursion boundary
    if (end - begin == 1)
        return:
    \overline{//} 1. Divide
    auto mid = begin + ((end - begin) >> 1); // equal to
        begin + (end - begin)/2
    vector < int > left (begin, mid), right (mid, end);
    // 2. Conquer
    merge sort(left.begin(), left.end());
    merge sort(right.begin(), right.end());
    // 3. Combine
    merge(left.begin(), left.end(), right.begin(),
        right.end(), begin);
```

std::partition()

```
int pivot;
bool cmp_partition(int x) { return x < pivot; }
void quicksort (vector < int >:: iterator begin,
   vector < int > :: iterator end)
    // 0. recursion boundary
    if (end - begin \ll 1)
        return:
    // 1. Divide
    pivot = *(end - 1);
    auto itr = partition (begin, end -1, cmp_partition);
    swap(*itr, *(end - 1));
    // 2. Conquer
    quicksort (begin, itr);
    quicksort(itr + 1, end);
    // 3. Combine
    // Nothing to do
```

std::nth_element()

```
nth\_element(v.begin(), v.begin() + 2, v.end());
cout << "\n2-nd_element_(0-indexed)_is_" <<
   *(v.begin() + 2) << '\n';
```

std::sort()

```
sort(v.begin(), v.end());
for (const int &i : v)
    cout << 'u' << i;</pre>
```

31/33

std::sort() with std::string and other...

```
string s = "hello_world_QWERIY_QAZ_QSC_QQ_wasd";
cout << '\n' + s << '\n';
sort(s.begin(), s.end());
cout << s << '\n';</pre>
```

std::sort() using custom comparator

```
bool cmp_sort(int lhs, int rhs) // result be like: {1,
   3, 5, 4, 2, 0
    if (lhs & 1 ^ rhs & 1) // if l\%2 != r\%2
        return lhs & 1; // return true if l is odd
            otherwise return false
    else
        return lhs & 1 ? lhs < rhs : lhs > rhs; // if l,
            r are both odd then return l < r else return
            l > r
    sort(v.begin(), v.end(), cmp sort);
    for (const int &i : v)
        cout << ',,' << i:
```