Sorting

Gleb Lobanov

Current Workplace – B2B YANGO

Tutoring – MIPT, Tinkoff,
olympiad schools and so on

Tg/discord - glebodin,
if you need something special – feel free to ask

How your mark is computed:

$$M_{\text{final}} = \min(5, round(0.7 \cdot M_{\text{practice}} + 0.3 \cdot M_{\text{exam}} + M_{\text{bonuses}} \cdot 0.1))$$

$M_{hw} = (\sum_{i=1}^{n} M_{hw_i})/n$, where n is the number of contests that we will have.

$$M_{hw_i} = 5 \cdot \sqrt{(\text{you solved in the } hw_i)/(\text{problems in the } hw_i)}$$

$$M_{\text{exam}} = 0.4 \cdot M_{\text{midterm}} + 0.6 \cdot M_{\text{final exam}}$$

$M_{iw} = (\sum_{i=1}^{n} M_{iw_i})/n$, where n is the number of tests that we will have

$$M_{iw_i} = 5 \cdot \sqrt{(\text{you solved in the } iw_i)/(\text{problems in the } iw_i)}$$

$$M_{\text{final exam}} = 5 \cdot \sqrt{(\text{you solved in the final exam})/(\text{problems in the finam exam})}$$

M_{bonuses}may be given on various unpredictable special occasions. Almost never is neg

Motivation

➔Better understanding of standard utilities

➔Code optimization

➔Understanding of useful algorithms

What pitfalls can you meet if you skip this course?

- ➔ Wrong understanding of asymptotics
- ➔ Wrong usages of structs
- ➔ Failing algo sections in interview

Why O-notation

- Different programming languages
- Different processor architecture
- Hard to measure time

Python

fork скачать

```
1. a = []
2. for i in range(1000000):
3.     a.append(i)
4. sum = 0
5. for i in a:
6.     sum += i
7. print(sum)
```

#stdin #stdout 0.33s 49220KB

stdin

```
Standard input is empty
```

stdout

```
499999500000
```

C++

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> a;
    for (int i = 0; i < 1000000; i++) {
        a.push_back(i);
    }
    long long sum = 0;
    for (auto elem : a) {
        sum += elem;
    }
    cout << sum;
}
```

#stdin #stdout 0.01s 7872KB

stdin

```
Standard input is empty
```

stdout

```
499999500000
```

Not all CPU operations are created equal

ithare.com

Operation	Cost in CPU Cycles
“Simple” register-register op (ADD,OR,etc.)	<1
Memory write	~1
Bypass delay: switch between integer and floating-point units	0-3
“Right” branch of “if”	1-2
Floating-point/vector addition	1-3
Multiplication (integer/float/vector)	1-7
Return error and check	1-7
L1 read	3-4
L2 read	10-12
“Wrong” branch of “if” (branch misprediction)	10-20
Floating-point division	10-40
128-bit vector division	10-70
C function direct call	15-30
Integer division	15-40
C function indirect call	20-50
C++ virtual function call	30-60
L3 read	30-70
Main RAM read	100-150
NUMA: different-socket L3 read	100-300
Allocation+deallocation pair (small objects)	200-500
NUMA: different-socket main RAM read	300-500
Kernel call	1000-1500
Thread context switch (direct costs)	2000
C++ Exception thrown+caught	5000-10000
Thread context switch (total costs, including cache invalidation)	10000 - 1 million

10^0 10^1 10^2 10^3 10^4 10^5 10^6

Distance which light travels while the operation is performed

Time

How to measure time

- Maybe too big
- Maybe too small
- Hard to measure time

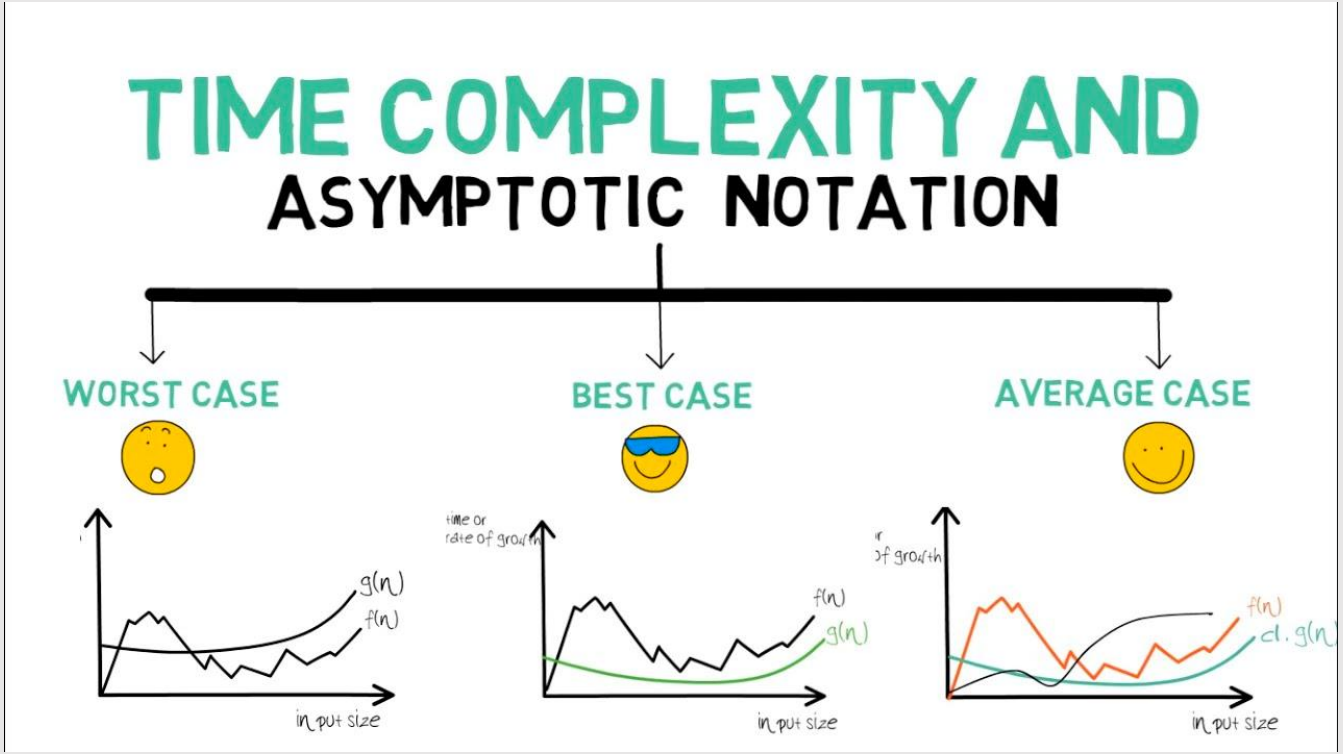
Assumptions

- Some operations are elementary
- We can’t do multiple operations in one operation

Elementary operations in O-notaion

- Bit operations(bitwise and, bitwise or, …);
- arithmetic operations (add, multiply, divide, …);
- some comparison operations(compare, greater, less, ..);
- some logical operations(and, or, …);
- some memory operations(take from memory, put to memory, …);
- some IO operations(read, write, …);
- If, exception and more other.

What is Asymptotic



```
for (int i = 0; i < n; i++) {
```

n

```
for i in range(n):
```

n

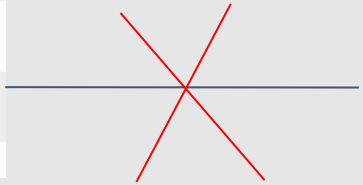
```python
for i in range(n):
    for j in range(n)
```

n^2

```java
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
```

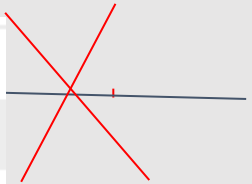
n^2

```
for i in range(n):
    for j in range(n - i * i):
```



n^2

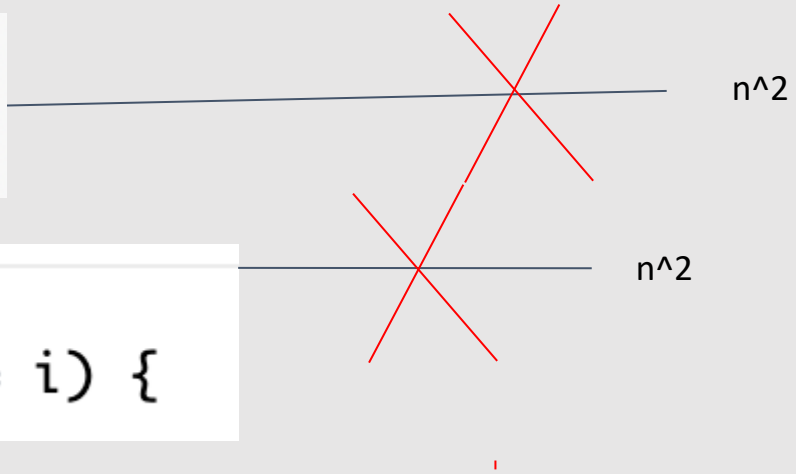
```
for (int i = 1; i <= n; i++) {
    for (int j = 0; j < n - i * i; j++) {
```



n^2

```python
for i in range(1, n + 1):
    for j in range(0, n, i):
```

```java
for (int i = 1; i <= n; i++) {
    for (int j = 0; j <= n; j += i) {
```



what is asymptotics(strict)

$$\mathrm{f(x)} \approx O(g(x)) \iff \forall x > C_1 : \frac{f(x)}{g(x)} \leq C_2$$

How to calculate

- I’ve already seen something similar
- Eazy calculation
- Make some assumptions
- Calculate some complex sum

```cpp
int minn = 0;
for (int i = 0; i < n; i++) {
    minn = min(minn, a[i]);
}
```

n

```python
minn = 0
for i in a:
    if a[i] < minn:
        minn = a[i]
```

n

```
int maxx = 0;
for (int i = 0; i < n; i++) {
    maxx = max(maxx, a[i]);
}
```

n

```
maxx = 0
for i in a:
    if a[i] > maxx:
        maxx = a[i]
```

n

Eazy calculation

```python
log_2 = 0
while n > 1:
    n = n // 2
    log_2 += 1
print(log_2)
```

```cpp
int log_2 = 0;
while (n > 1) {
    n /= 2;
    log_2++;
}
```

log(n)

```python
for i in range(n):
    for j in range(n - i * i):
```

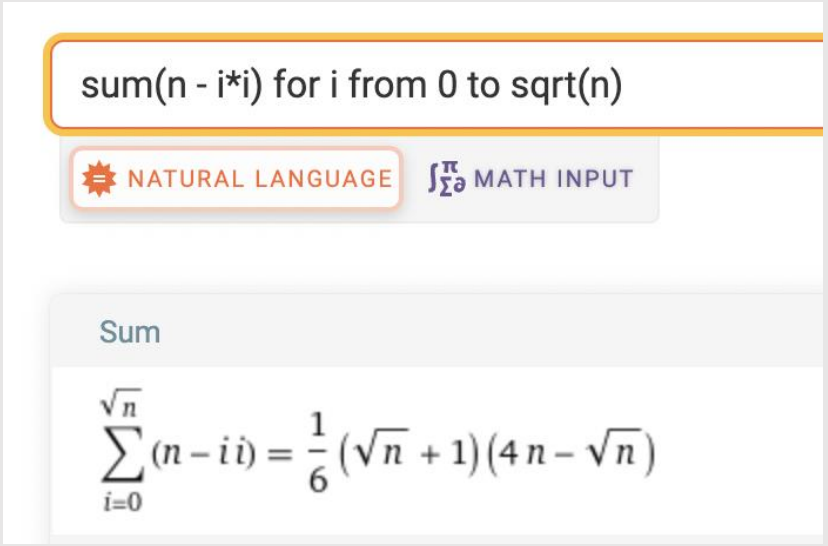
<= O(n^2)

```java
for (int i = 1; i <= n; i++) {
    for (int j = 0; j < n - i * i; j++) {
```

<= O(n^2)

```java
for (int i = 1; i <= n; i++) {
    for (int j = 0; j < n - i * i; j++) {
```

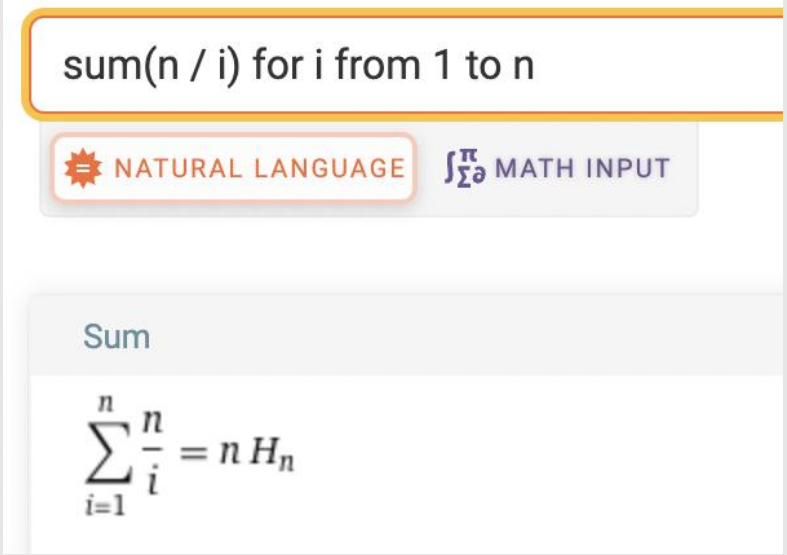
```python
for i in range(n):
    for j in range(n - i * i):
```



Example

```java
for (int i = 1; i <= n; i++) {
    for (int j = 0; j <= n; j += i) {
```

```python
for i in range(1, n + 1):
    for j in range(0, n, i):
```



Growth rate [edit]

These numbers grow very slowly, with logarithmic growth, as can be seen from the integral test.[15] More precisely, by the Euler–Maclaurin formula,

$$H_n = \ln n + \gamma + \frac{1}{2n} - \varepsilon_n$$

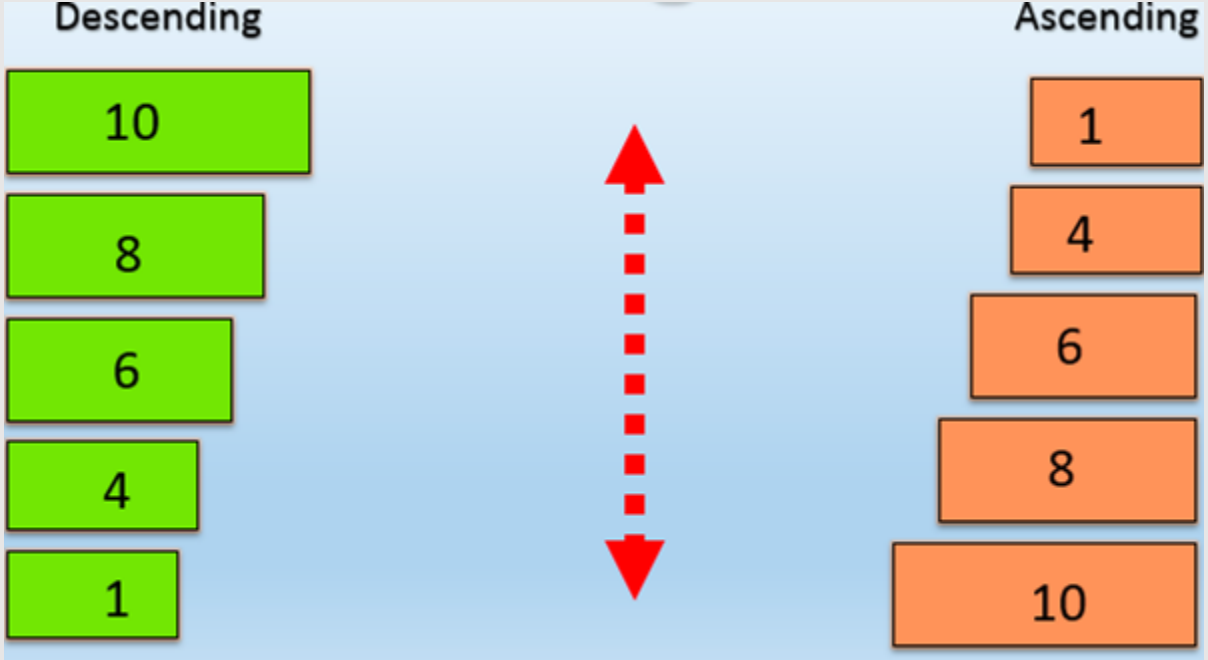
where $\gamma \approx 0.5772$ is the Euler–Mascheroni constant and $0 \leq \varepsilon_n \leq 1/8n^2$ which approaches 0 as n goes to infinity.[16]

```
for i in range(n):
    for j in range(n - i * i):
```

$n\sqrt{n}$

```
for i in range(1, n + 1):
    for j in range(0, n, i):
```

$n \log(n)$


Descending
10
8
6
4
1
Ascending
1
4
6
8
10

Sorting

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
    int a[10];
    for (int i = 0; i < 10; i++) {
        cin >> a[i];
    }

    sort(a, a + 10);

    for (int i = 0; i < 10; i++) {
        cout << a[i] << " ";
    }
}
```

#stdin #stdout 0.01s 5544KB

stdin

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

stdout

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

```python
a = input().split()

a.sort()

print(a)
```

#stdin #stdout 0.03s 9656KB

stdin

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

stdout

```
['1', '1', '1', '2', '2', '2', '3', '3', '4', '5']
```

```
1. a = input().split()
2.
3. a.sort()
4.
5. print(a)
```

#stdin #stdout 0.03s 9648KB

stdin

```
15 1 23 2 33 3
```

stdout

```
['1', '15', '2', '23', '3', '33']
```

```
1. a = list(map(int, input().split()))
2.
3. a.sort()
4.
5. print(a)
```

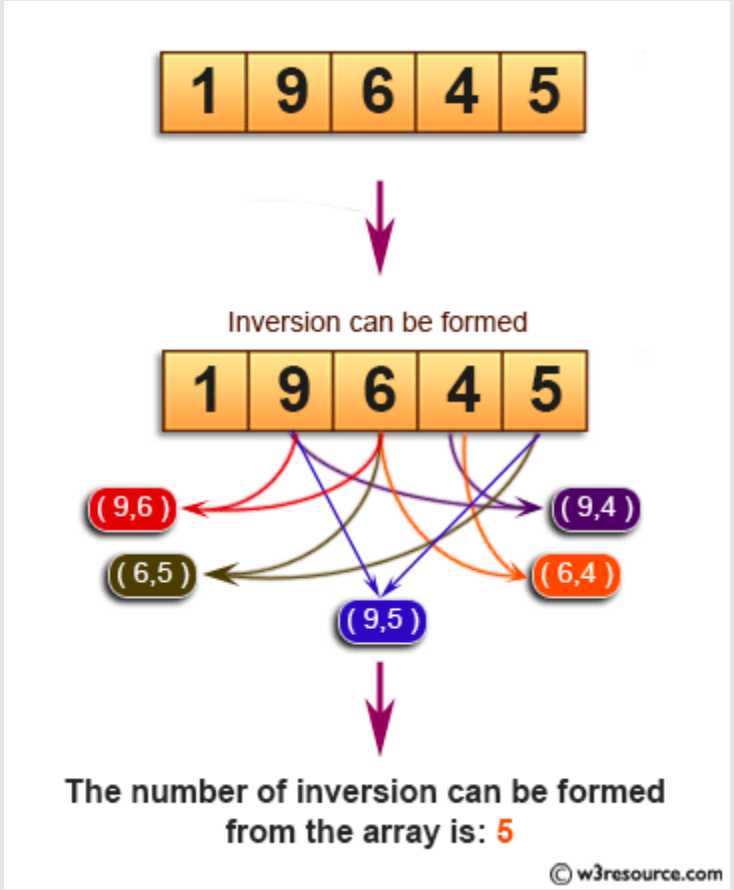
#stdin #stdout 0.04s 9836KB

stdin

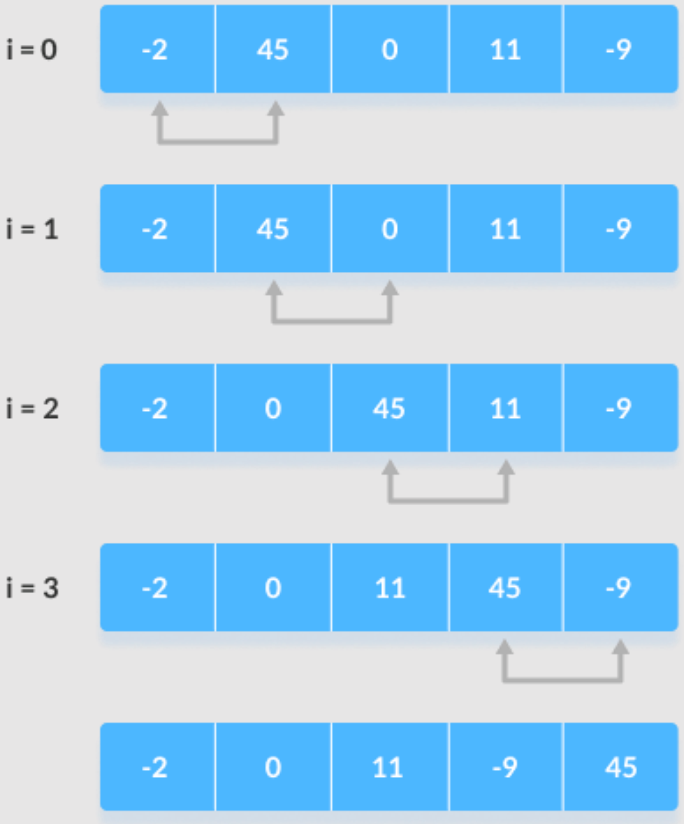
```
15 1 23 2 33 3
```

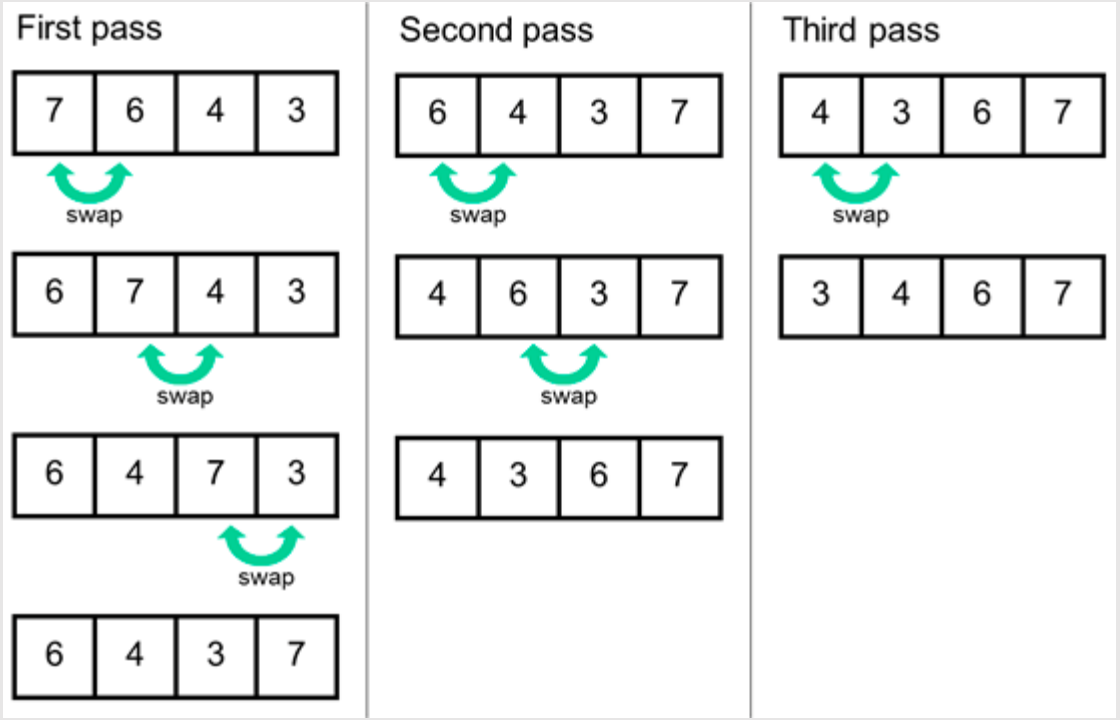
stdout

```
[1, 2, 3, 15, 23, 33]
```


1 9 6 4 5
Inversion can be formed
1 9 6 4 5
(9,6)
(9,4)
(6,5)
(6,4)
(9,5)
The number of inversion can be formed from the array is: 5
© w3resource.com

step = 0




First pass
7 6 4 3
swap
6 7 4 3
swap
6 4 7 3
swap
6 4 3 7
Second pass
6 4 3 7
swap
4 6 3 7
swap
4 3 6 7
Third pass
4 3 6 7
swap
3 4 6 7

[Visualization]

Worst-case performance	$O(n^2)$ comparisons, $O(n^2)$ swaps
Best-case performance	$O(n)$ comparisons, $O(1)$ swaps
Average performance	$O(n^2)$ comparisons, $O(n^2)$ swaps
Worst-case space complexity	$O(n)$ total, $O(1)$ auxiliary

```cpp
#include <iostream>

using namespace std;
const int LEN = 10;

int main() {
    int a[LEN];
    for (int i = 0; i < LEN; i++) {
        cin >> a[i];
    }

    for (int amount_sifting = 0; amount_sifting < LEN; amount_sifting++) {
        for (int elem = 0; elem < LEN - 1; elem++) {
            if (a[elem] > a[elem + 1]) {
                swap(a[elem], a[elem + 1]);
            }
        }
    }

    for (int i = 0; i < LEN; i++) {
        cout << a[i] << " ";
    }
}
```

Success #stdin #stdout 0.01s 5312KB

stdin

```
4 3 2 1 1 2 3 4 4 3
```

stdout

```
1 1 2 2 3 3 3 4 4 4
```

fork скачать

```
1. a_strings = input().split()
2. a = list(map(int, a_strings))
3. 
4. for amount_sifting in range(len(a)):
5.     for index in range(len(a) - 1):
6.         if a[index] > a[index + 1]:
7.             a[index], a[index + 1] = a[index + 1], a[index]
8. 
9. print(a)
```

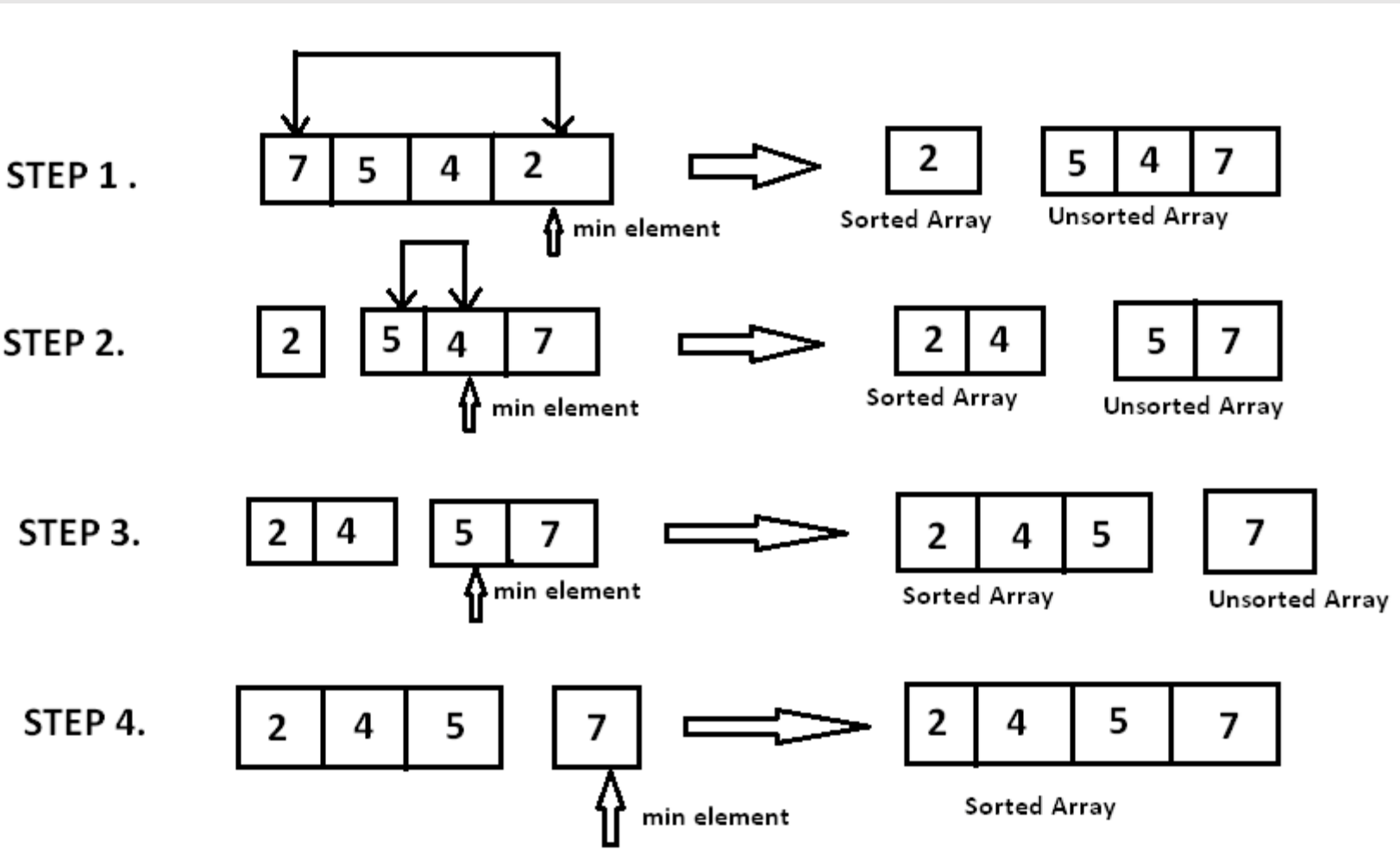
#stdin #stdout 0.04s 9696KB

stdin

```
11 1 2 3 22
```

stdout

```
[1, 2, 3, 11, 22]
```


STEP 1 .
7
5
4
2
min element
2
Sorted Array
5
4
7
Unsorted Array
STEP 2.
2
5
4
7
min element
2
4
Sorted Array
5
7
Unsorted Array
STEP 3.
2
4
5
7
min element
2
4
5
Sorted Array
7
Unsorted Array
STEP 4.
2
4
5
7
min element
2
4
5
7
Sorted Array

[Visualization]

Worst-case performance	$O(n^2)$ comparisons, $O(n)$ swaps
Best-case performance	$O(n^2)$ comparisons, $O(1)$ swap
Average performance	$O(n^2)$ comparisons, $O(n)$ swaps
Worst-case space complexity	$O(1)$ auxiliary

```cpp
#include <iostream>

using namespace std;
const int LEN = 6;

int main() {
    int a[LEN] = {11, 2, 1, 3, 33, 22};
    for (int index = 0; index < LEN; index++) {
        int min_index = index;
        for (int j = index + 1; j < LEN; j++) {
            if (a[j] < a[min_index]) {
                min_index = j;
            }
        }
        swap(a[index], a[min_index]);
    }
    for (int i = 0; i < LEN; i++) {
        cout << a[i] << " ";
    }
}
```

#stdin #stdout 0.01s 5368KB

stdin

```
11 2 1 3 33 22
```

stdout

```
1 2 3 11 22 33
```

```python
a = list(map(int, input().split()))

for index in range(len(a)):
    min_index = index
    for j in range(index + 1, len(a)):
        if a[j] < a[min_index]:
            min_index = j
    (a[index], a[min_index]) = (a[min_index], a[index])

print(a)
```

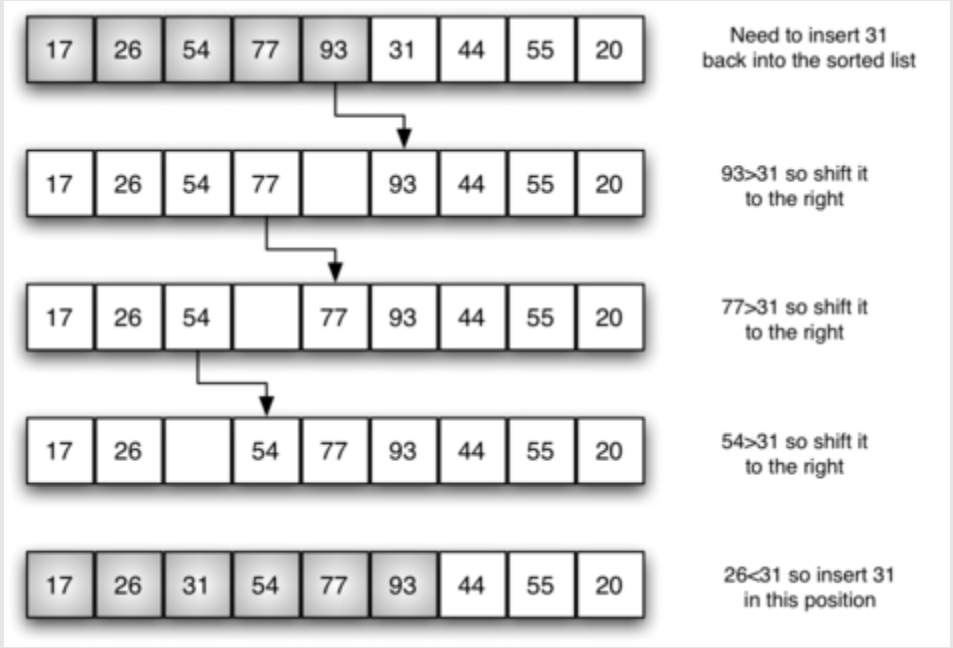
#stdin #stdout 0.03s 9676KB

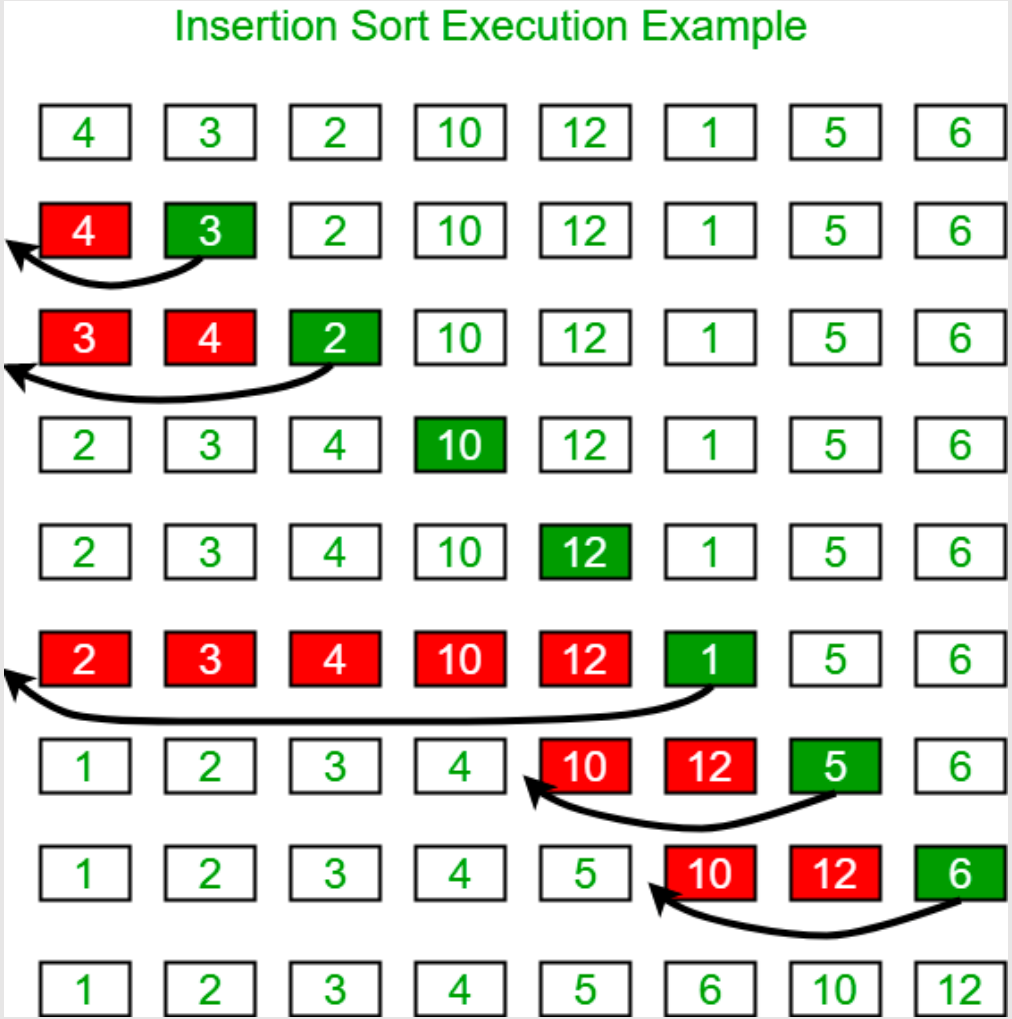
stdin

```
11 2 1 3 33 22
```

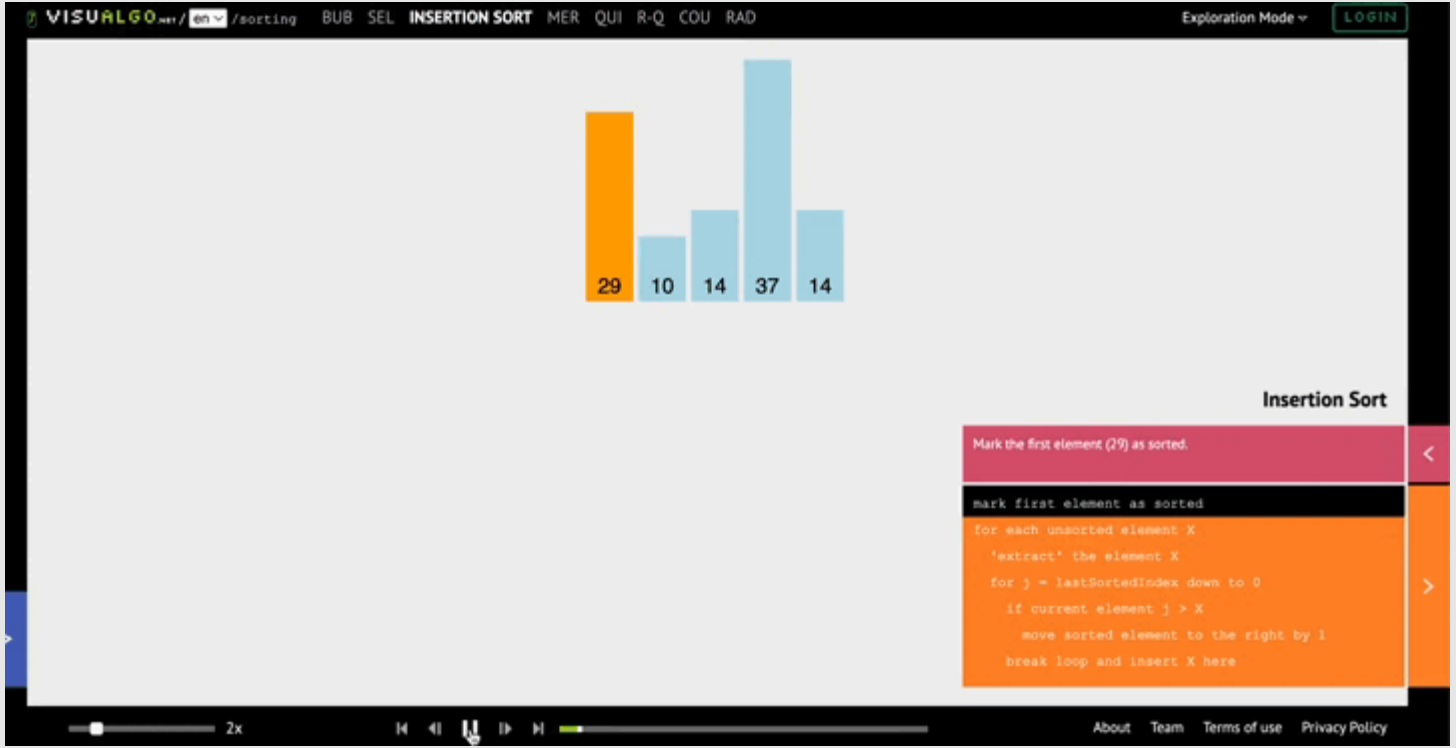
stdout

```
[1, 2, 3, 11, 22, 33]
```


17 26 54 77 93 31 44 55 20
Need to insert 31 back into the sorted list
17 26 54 77 93 44 55 20
93>31 so shift it to the right
17 26 54 77 93 44 55 20
77>31 so shift it to the right
17 26 54 77 93 44 55 20
54>31 so shift it to the right
17 26 31 54 77 93 44 55 20
26<31 so insert 31 in this position


Insertion Sort Execution Example
4 3 2 10 12 1 5 6
4 3 2 10 12 1 5 6
3 4 2 10 12 1 5 6
2 3 4 10 12 1 5 6
2 3 4 10 12 1 5 6
2 3 4 10 12 1 5 6
1 2 3 4 10 12 5 6
1 2 3 4 5 10 12 6
1 2 3 4 5 6 10 12

Visualization



Worst-case performance	$O(n^2)$ comparisons and swaps
Best-case performance	$O(n)$ comparisons, $O(1)$ swaps
Average performance	$O(n^2)$ comparisons and swaps
Worst-case space complexity	$O(n)$ total, $O(1)$ auxiliary

```cpp
#include <iostream>

using namespace std;
const int LEN = 6;

int main() {
    int a[LEN] = {11, 2, 1, 3, 33, 22};
    for (int i = 1; i < LEN; i++) {
        int key = a[i];
        int j = i - 1;
        while (j >= 0 && a[j] > key) {
            a[j + 1] = a[j];
            j = j - 1;
        }
        a[j + 1] = key;
    }
    for (int i = 0; i < LEN; i++) {
        cout << a[i] << " ";
    }
}
```

#stdin #stdout 0.01s 5360KB

stdin

```
11 2 1 3 33 22
```

stdout

```
1 2 3 11 22 33
```

```python
a = list(map(int, input().split()))

for index in range(1, len(a)):
    tmp = a[index]
    j = index - 1
    while (j >= 0 and tmp < a[j]):
        a[j + 1] = a[j]
        j = j - 1
    a[j + 1] = tmp
print(a)
```

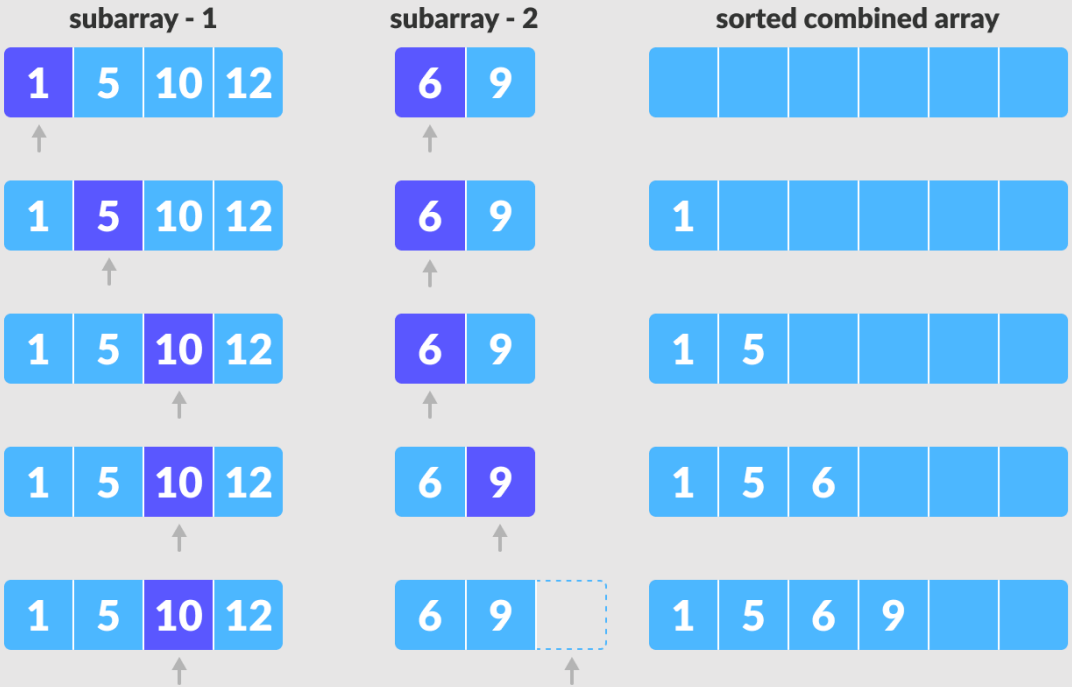
#stdin #stdout 0.03s 9704KB

stdin

```
11 2 1 3 33 22
```

stdout

```
[1, 2, 3, 11, 22, 33]
```

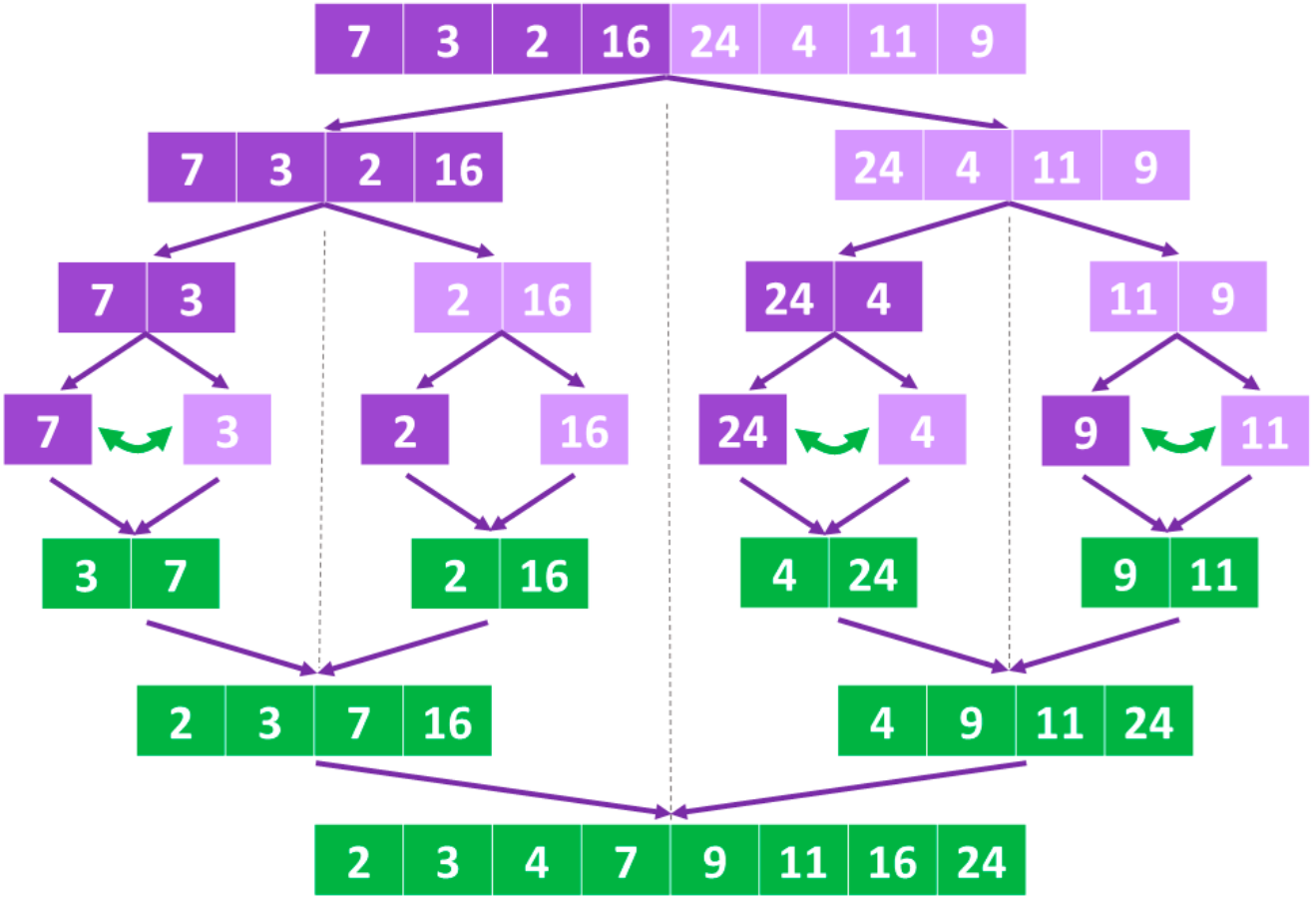


Since there are no more elements remaining in the second array, and we know that both the arrays were sorted when we started, we can copy the remaining elements from the first array directly.

```python
def merge(left, right):
    result = []
    i, j = 0, 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    while i < len(left):
        result.append(left[i])
        i += 1
    while j < len(right):
        result.append(right[j])
        j += 1
    return result

```

```cpp
void merge(vector<int> &a, int l, int mid, int r) {
    vector<int> result;
    int i = l, j = mid;

    while (i < mid && j < r) {
        if (a[i] < a[j]) {
            result.push_back(a[i]);
            i++;
        }
        else {
            result.push_back(a[j]);
            j++;
        }
    }
    while (i < mid) {
        result.push_back(a[i]);
        i++;
    }
    while (j < r) {
        result.push_back(a[j]);
        j++;
    }

    for (int i = l; i < r; i++) {
        a[i] = result[i - l];
    }
}
```

Merge Sort



Step 1:
Split sub-lists in two until you reach pair of values.

Step 3:
Sort/swap pair of values if needed.

Step 4:
Merge and sort sub-lists and repeat process till you merge to the full list.

Visualization

Worst-case performance	$O(n \log n)$
Best-case performance	$\Omega(n \log n)$ typical, $\Omega(n)$ natural variant
Average performance	$\Theta(n \log n)$
Worst-case space complexity	$O(n)$ total with $O(n)$ auxiliary, $O(1)$ auxiliary with linked lists[1]

Why O(nlogn)

- Each time we divide our array in two halves, so we will have O(log(n)) layers
- Merge function works in linear time
- In each layer we have work time = O(sum(|block_1|)) = O(n)
- Final time = O(n) * O(log(n)) = O(nlog(n))

```
18.
19. def merge_sort(a):
20.     if len(a) < 2:
21.         return a[:]
22.     else:
23.         middle = int(len(a) / 2)
24.         left = merge_sort(a[:middle])
25.         right = merge_sort(a[middle:])
26.         return merge(left, right)
27.
28. a = list(map(int, input().split()))
29.
30. print(merge_sort(a))
31.
```

#stdin #stdout 0.05s 9768KB

stdin

```
11 2 1 3 33 22
```

stdout

```
[1, 2, 3, 11, 22, 33]
```

```cpp
void merge_sort(vector<int> &a, int l, int r) {
    if (l != r - 1) {
        int mid = (l + r) / 2;
        merge_sort(a, l, mid);
        merge_sort(a, mid, r);
        merge(a, l, mid, r);
    }
}

int main() {
    vector<int> a = {11, 2, 3, 22, 1, 33};

    merge_sort(a, 0, a.size());

    for (int i = 0; i < a.size(); i++) {
        cout << a[i] << " ";
    }
    return 0;
}
```

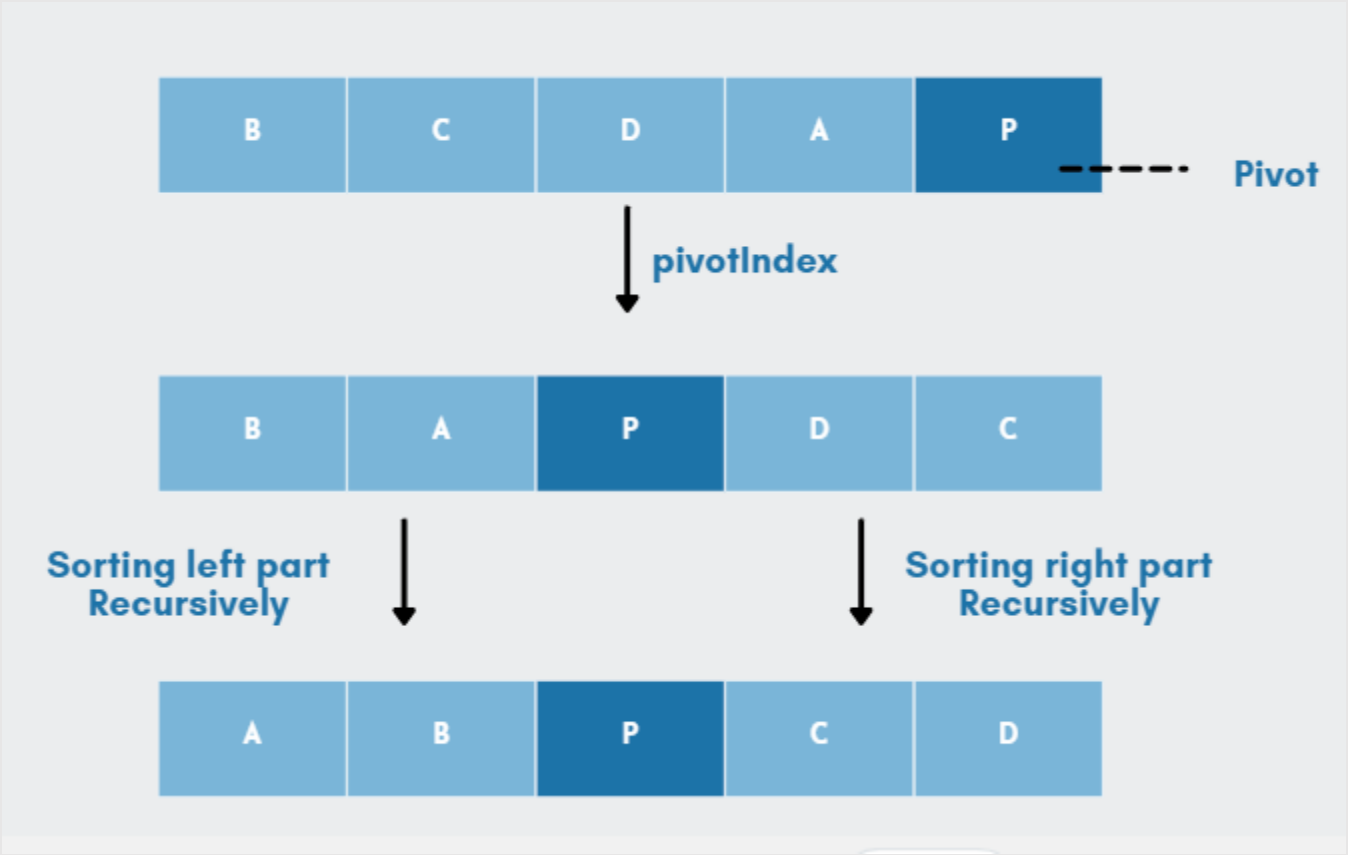
#stdin #stdout 0.01s 5368KB

stdin

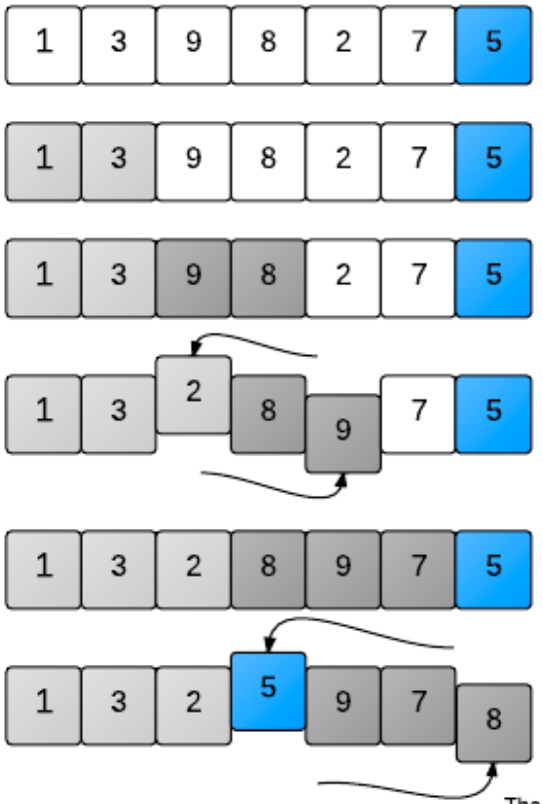
```
Standard input is empty
```

stdout

```
1 2 3 11 22 33
```

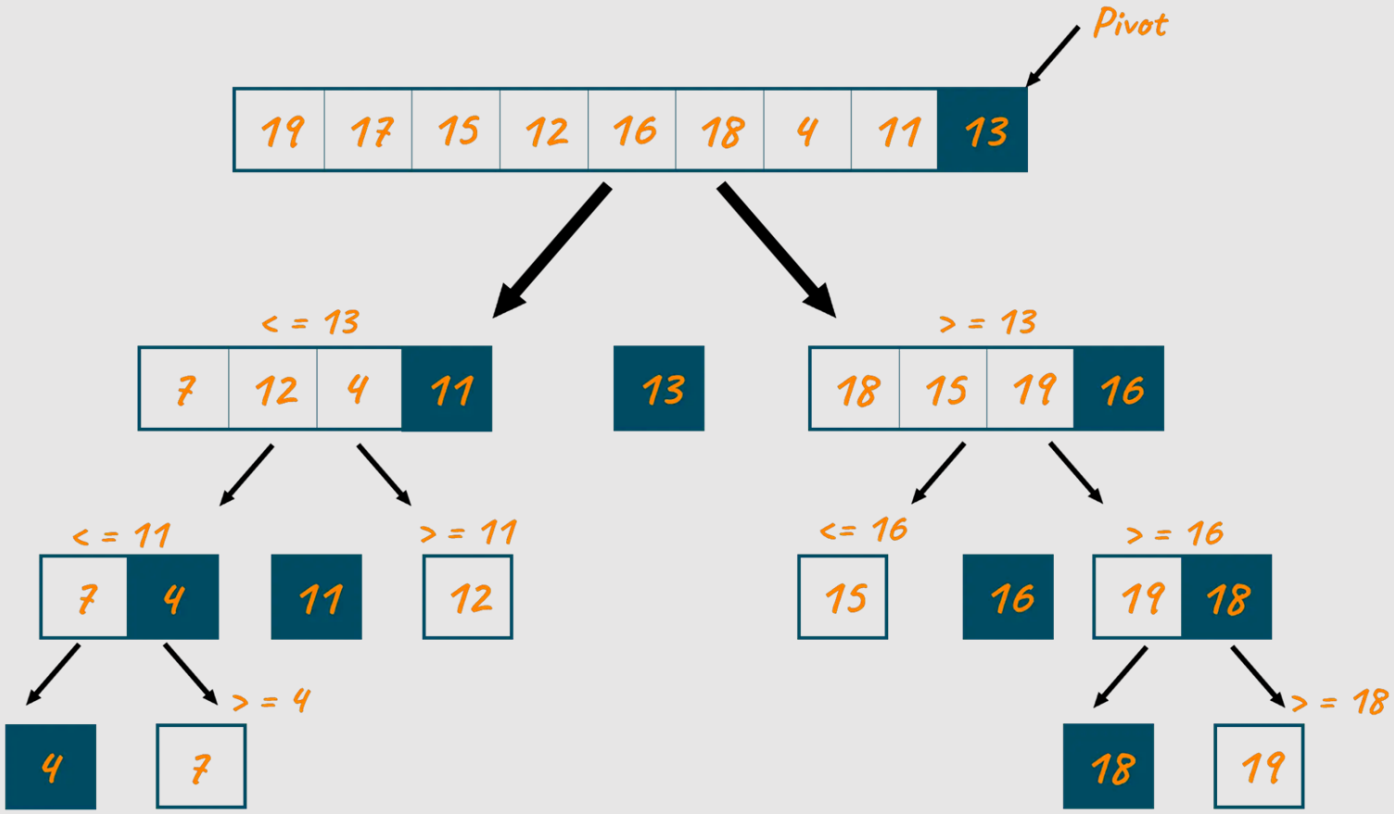

B
C
D
A
P
Pivot
pivotIndex
B
A
P
D
C
Sorting left part
Recursively
Sorting right part
Recursively
A
B
P
C
D

Partitioning an array



```
def partition(a, l, r):
    pivot = a[r];
    pIndex = l;

    for i in range(l, r):
        if (a[i] <= pivot):
            a[i], a[pIndex] = a[pIndex], a[i]
            pIndex += 1
    a[pIndex], a[r] = a[r], a[pIndex]

    return pIndex;
```

```
int partition(vector<int> &a, int l, int r) {
    int pivot = a[r];
    int pIndex = l;

    for (int i = l; i < r; i++) {
        if (a[i] <= pivot) {
            swap(a[i], a[pIndex]);
            pIndex++;
        }
    }
    swap(a[pIndex], a[r]);

    return pIndex;
}
```


Pivot
19 17 15 12 16 18 4 11 13
< = 13
7 12 4 11
13
> = 13
18 15 19 16
< = 11
7 4
11
12
> = 11
<= 16
15
16
19 18
> = 16
4
7
> = 4
18
19
> = 18

[Visualization]

```cpp
void quicksort(vector<int> &a, int start, int end) {
    if (start >= end) {
        return;
    }

    int pivot = partition(a, start, end);
    quicksort(a, start, pivot - 1);
    quicksort(a, pivot + 1, end);
}

int main() {
    vector<int> a = {11, 2, 3, 22, 1, 33, 5};

    quicksort(a, 0, a.size() - 1);

    for (int i = 0; i < a.size(); i++) {
        cout << a[i] << " ";
    }
    return 0;
}
```

#stdin #stdout 0.01s 5528KB

stdin

```
Standard input is empty
```

stdout

```
1 2 3 5 11 22 33
```

```
13. def quicksort(a, start, end):
14.     if (start >= end):
15.         return
16.
17.     pivot = partition(a, start, end)
18.     quicksort(a, start, pivot - 1)
19.     quicksort(a, pivot + 1, end)
20.
21. a = list(map(int, input().split()))
22.
23. quicksort(a, 0, len(a) - 1);
24.
25. print(a)
```

#stdin #stdout 0.03s 9708KB

stdin

```
2 33 1 3 4 5 12
```

stdout

```
[1, 2, 3, 4, 5, 12, 33]
```

Pivot

- First element
- Last element
- 3-Median
- Random element
- Median

Improvements

- Equal elements
- Combination with other sorting methods
- Binary quicksort
- O(nlog(n))

How to break pivot

- First element - [1, 2, 3, 4, 5, 6, 7, …]
- Last element - [1, 2, 3, 4, 5, 6, 7, …]
- 3-Median - [1, 3, 5, 7, 8, 6, 4, 2]

Worst-case performance	$O(n^2)$
Best-case performance	$O(n \log n)$ (simple partition) or $O(n)$ (three-way partition and equal keys)
Average performance	$O(n \log n)$
Worst-case space complexity	$O(n)$ auxiliary (naive) $O(\log n)$ auxiliary (Hoare 1962)

A = [2, 1, 3, 4, 8, 7, 6, 9, 5]
A_set = {1, 2, 3, 4, 5, 6, 7, 8, 9}

{1, 2, 3, 4, 5, 6, 7, 8, 9}

good subarray to choose pivot

Why?

- If we choose pivot from this subarray, than we have two parts and the biggest one <= ¾ * n
- If we choose pivot from this subarray, working time - O(n) * log_{4/3}(n)
- Final working time - O(n) * log_{4/3}(n) * 2 = O(nlog(n))

Let C(n) - working time for input of length n

$C(n) = n - 1 + \frac{1}{n}\sum_{i=0}^{n-1}(C(i) + C(n-i-1))$ - because we divide to 2 arrays of size i and $n-i-1$

$C(n) = n - 1 + \frac{2}{n}\sum_{i=0}^{n-1} C(i)$ - just split sum to 2 sums

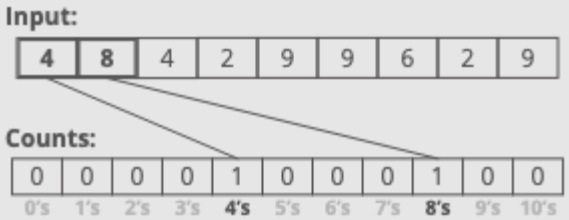
$n \cdot C(n) = n \cdot (n-1) + 2\sum_{i=0}^{n-1} C(i)$ - multiply both parts by n

$n \cdot C(n) - (n-1) \cdot C(n-1) = n \cdot (n-1) - (n-1) \cdot (n-2) + 2 \cdot C(n-1) \rightarrow n \cdot C(n) = (n+1) \cdot C(n-1) + 2n - 2$

$$\begin{aligned}
\frac{C(n)}{n+1} &= \frac{C(n-1)}{n} + \frac{2}{n+1} - \frac{2}{n(n+1)} \leq \frac{C(n-1)}{n} + \frac{2}{n+1} \\
&= \frac{C(n-2)}{n-1} + \frac{2}{n} - \frac{2}{(n-1)n} + \frac{2}{n+1} \leq \frac{C(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\
&\vdots \\
&= \frac{C(1)}{2} + \sum_{i=2}^{n} \frac{2}{i+1} \leq 2\sum_{i=1}^{n-1} \frac{1}{i} \approx 2\int_1^n \frac{1}{x}\mathrm{d}x = 2\ln n
\end{aligned}$$

Why O(nlogn) is the best

- We can sort only with comparisons
- If we did k comparisons, we have 2^k possible results
- We have n! possible permutations
- We need at least log_2(n!) comparisons to find sorted.
- log_2(n!) >= log_2((n / 2)^n/2) = log_2(n / 2) * n / 2 = (log_2(n) - 1) * n / 2 = O(nlogn)


Input:
4
8
4
2
9
9
6
2
9
Counts:
0
0
0
0
1
0
0
0
1
0
0
0's
1's
2's
3's
4's
5's
6's
7's
8's
9's
10's

Input Data

0	4	2	2	0	0	1	1	0	1	0	2	4	2

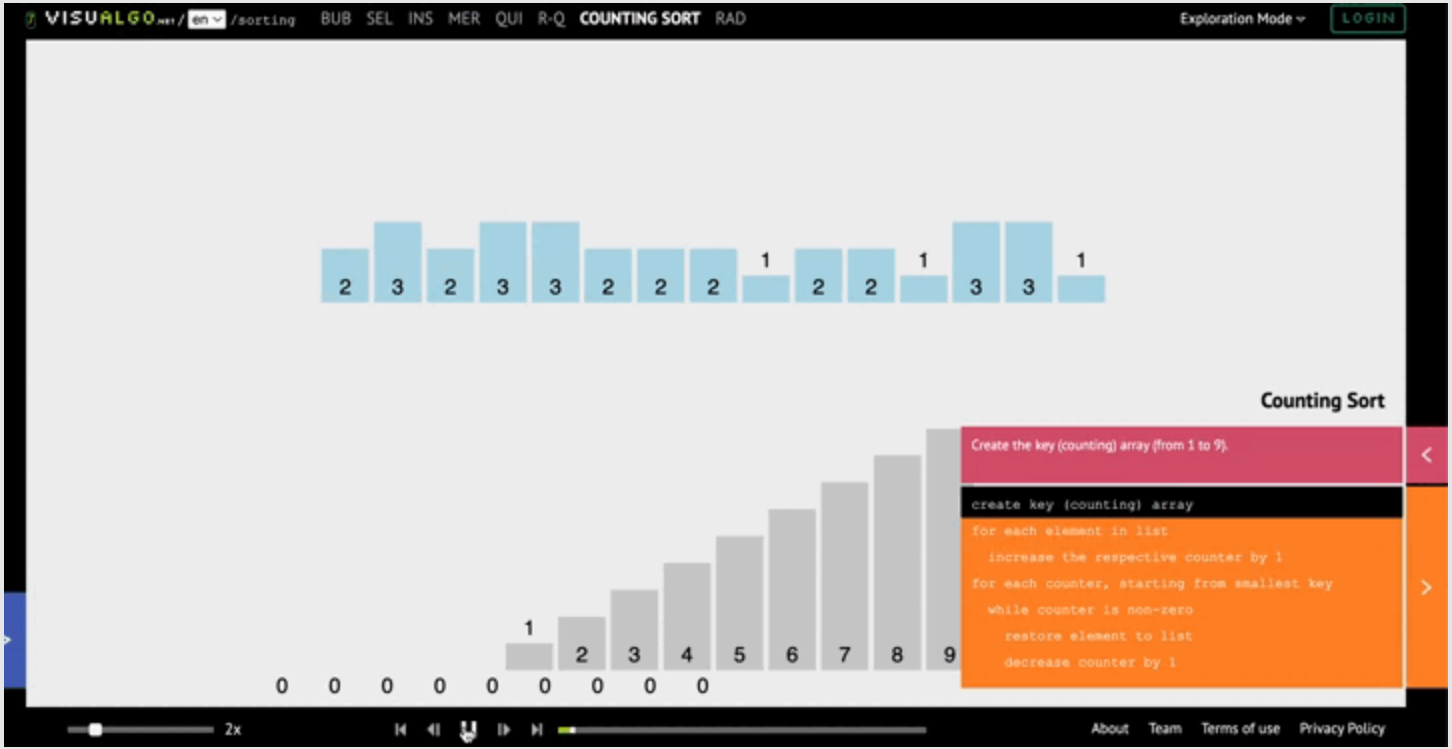
Count Array

0	1	2	3	4
5	3	4	0	2

Sorted Data

0	0	0	0	0	1	1	1	2	2	2	2	4	4

Visualization



Asymptotic

- O(n) - scanning of array
- O(MAX_NUM) - count array

```cpp
const int MAX_NUMBER = 5;
int count[MAX_NUMBER];

int main() {
    vector<int> a = {1, 2, 3, 2, 3, 1, 3, 2, 3, 4, 2, 3};

    for (int i = 0; i < a.size(); i++) {
        count[a[i]]++;
    }

    vector<int> sorted_a;
    for (int i = 0; i < MAX_NUMBER; i++) {
        cout << i << " " << count[i] << "\n";
        while (count[i] > 0) {
            sorted_a.push_back(i);
            count[i]--;
        }
    }

    for (int i = 0; i < sorted_a.size(); i++) {
        cout << sorted_a[i] << " ";
    }
    return 0;
}
```

#stdin #stdout 0.01s 5516KB

comments ()

stdin

copy

```
Standard input is empty
```

stdout

copy

```
0 0
1 2
2 4
3 5
4 1
1 1 2 2 2 2 3 3 3 3 3 4
```

```python
MAX_ELEM = 10

a = list(map(int, input().split()))

count = [0] * MAX_ELEM
for elem in a:
    count[elem] += 1

sorted_a = []

for i in range(MAX_ELEM):
    print(i, count[i])
    while count[i] > 0:
        sorted_a.append(i)
        count[i] -= 1

print(sorted_a)
```

#stdin #stdout 0.03s 9780KB

comments ()

stdin

copy

```
2 9 1 1 2 3 4 5 1 2 3 4 1 2 3 4 5 6 1 2 3 4 5 6 3 1 1 2
```

stdout

copy

```
0 0
1 7
2 6
3 5
4 4
5 3
6 2
7 0
8 0
9 1
[1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 6, 6, 9]
```

Stable Vs Unstable Sorts

Unsorted Array

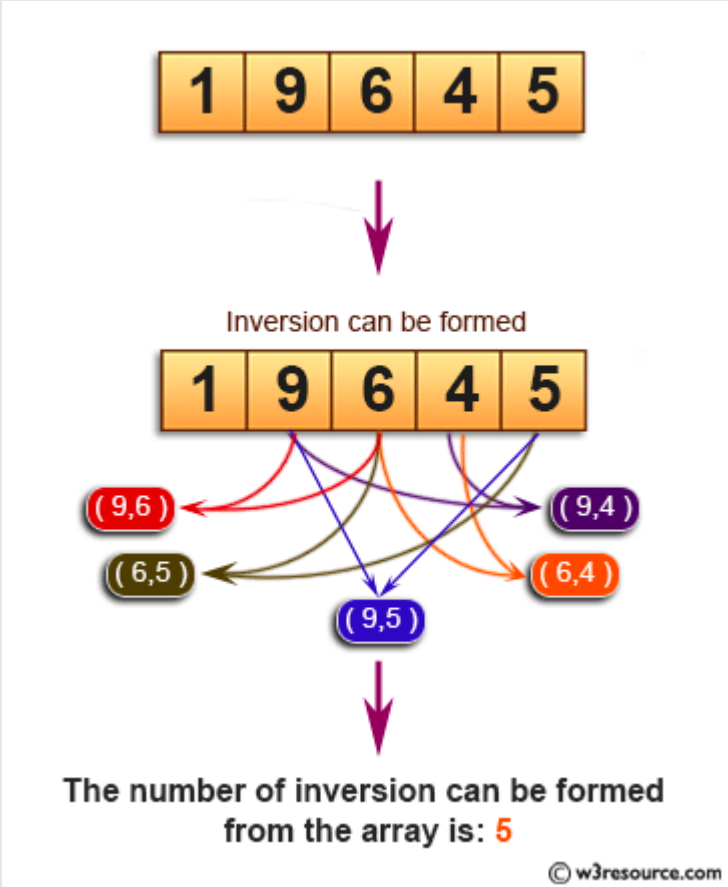
Stable Sort

Unstable Sort

© dizauvi.com

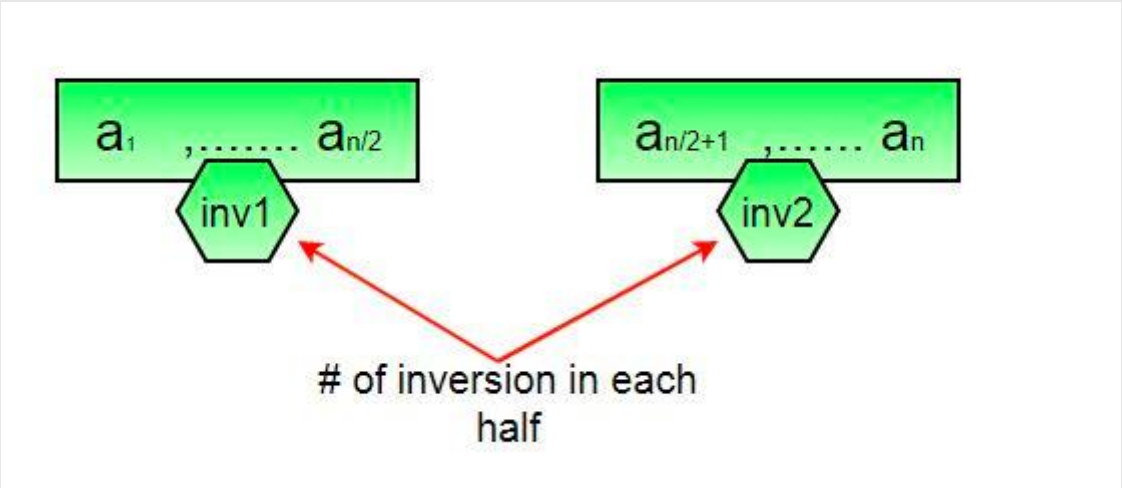
Sorting Algorithms	In - Place	Stable
Bubble Sort	Yes	Yes
Selection Sort	Yes	maybe
Insertion Sort	Yes	Yes
Quick Sort	Yes	No
Merge Sort	No (because it requires an extra array to merge the sorted subarrays)	Yes

Inversion amount

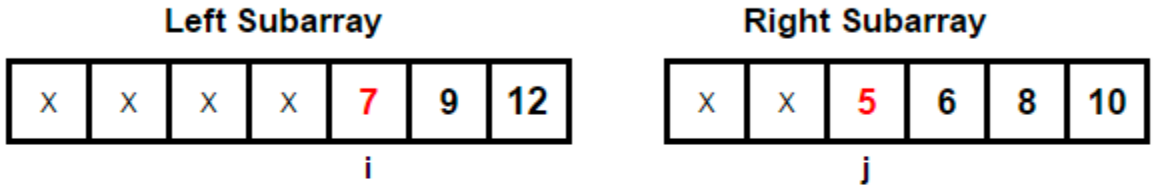


Inversion count

- We split the array into two parts and in each part we find the number of inversions.
- We find the number of inversions between the two sorted subarrays.

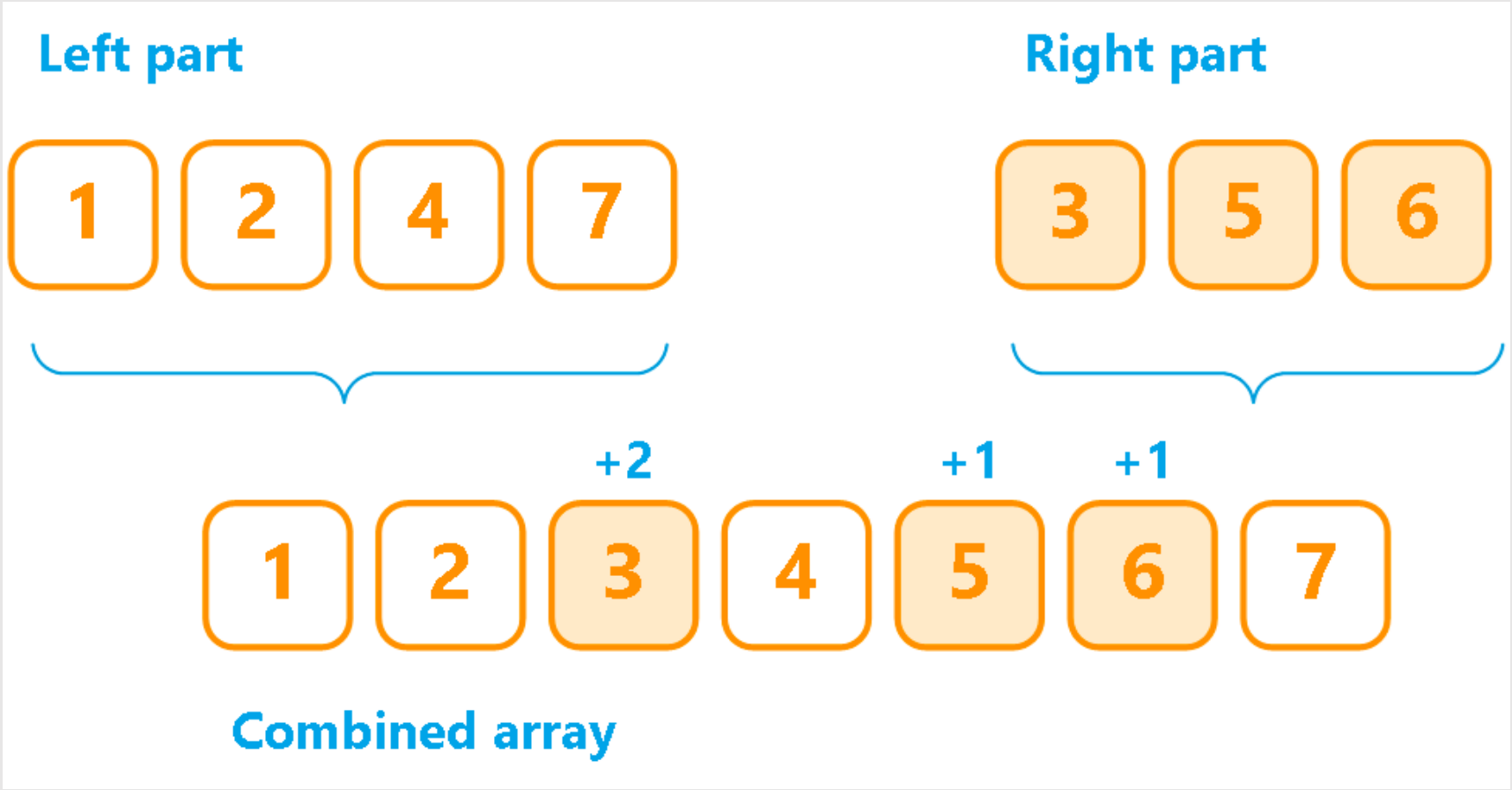

a_1 ,..... $a_{n/2}$
$a_{n/2+1}$,..... a_n
inv1
inv2
# of inversion in each half

Merge process



As 7 > 5, (7, 5) forms an inversion pair.

Also as left subarray is sorted, it is obvious that elements 9 and 12 will also form inversion with element 5. i.e (9, 5) and (12, 5). So we can say that for element 5, total number of inversions are 3 which is exactly equal to number of elements left in the left subarray.


Left part
Right part
1
2
4
7
3
5
6
+2
+1
+1
1
2
3
4
5
6
7
Combined array

```cpp
int merge(vector<int> &a, int l, int mid, int r) {
    vector<int> result;
    int i = l, j = mid;
    int count = 0;

    while (i < mid && j < r) {
        if (a[i] < a[j]) {
            result.push_back(a[i]);
            i++;
        }
        else {
            count += mid - i;
            result.push_back(a[j]);
            j++;
        }
    }
    while (i < mid) {
        result.push_back(a[i]);
        i++;
    }
    while (j < r) {
        result.push_back(a[j]);
        j++;
    }

    for (int i = l; i < r; i++) {
        a[i] = result[i - l];
    }
    return count;
}
```

```
37. int merge_sort(vector<int> &a, int l, int r) {
38.     if (l != r - 1) {
39.         int mid = (l + r) / 2;
40.         int count = merge_sort(a, l, mid);
41.         count += merge_sort(a, mid, r);
42.         count += merge(a, l, mid, r);
43.         return count;
44.     }
45.     return 0;
46. }
47.
48. int main() {
49.     vector<int> a = {11, 2, 3, 22, 1, 33};
50.
51.     cout << merge_sort(a, 0, a.size());
52.     return 0;
53. }
```

#stdin #stdout 0.01s 5448KB

stdin

```
Standard input is empty
```

stdout

```
6
```

```python
def merge(left, right):
    result = []
    i, j = 0, 0
    amount = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            amount += (len(left) - i)
            result.append(right[j])
            j += 1
    while i < len(left):
        result.append(left[i])
        i += 1
    while j < len(right):
        result.append(right[j])
        j += 1
    return (result, amount)

```

```python
def merge_sort(a):
    if len(a) < 2:
        return (a[:], 0)
    else:
        middle = int(len(a) / 2)
        (left, count_1) = merge_sort(a[:middle])
        (right, count_2) = merge_sort(a[middle:])
        (vec, count) = merge(left, right)
        return (vec, count + count_2 + count_1)

a = list(map(int, input().split()))

print(merge_sort(a))

```

#stdin #stdout 0.03s 9676KB

stdin

```
11 2 1 3 33 22
```

stdout

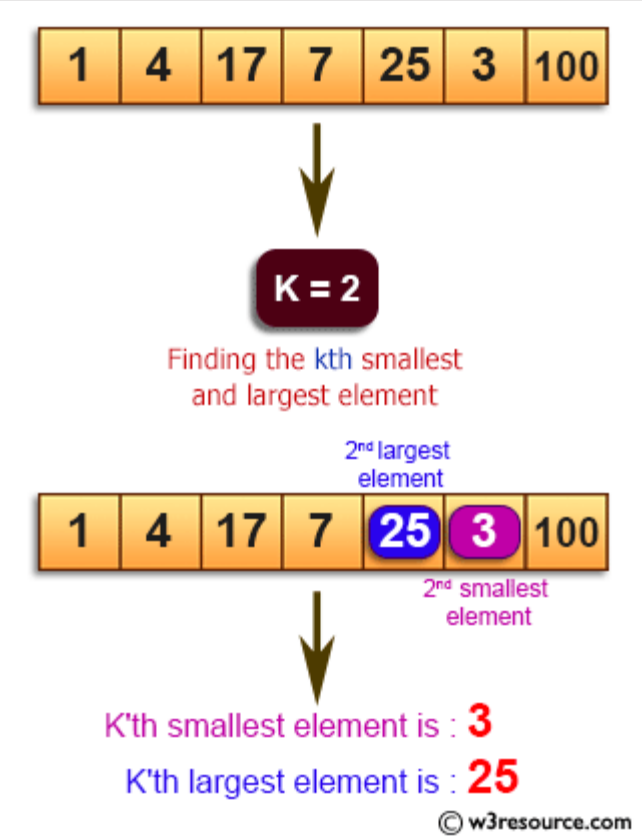
```
([1, 2, 3, 11, 22, 33], 5)
```

Merge sort tree

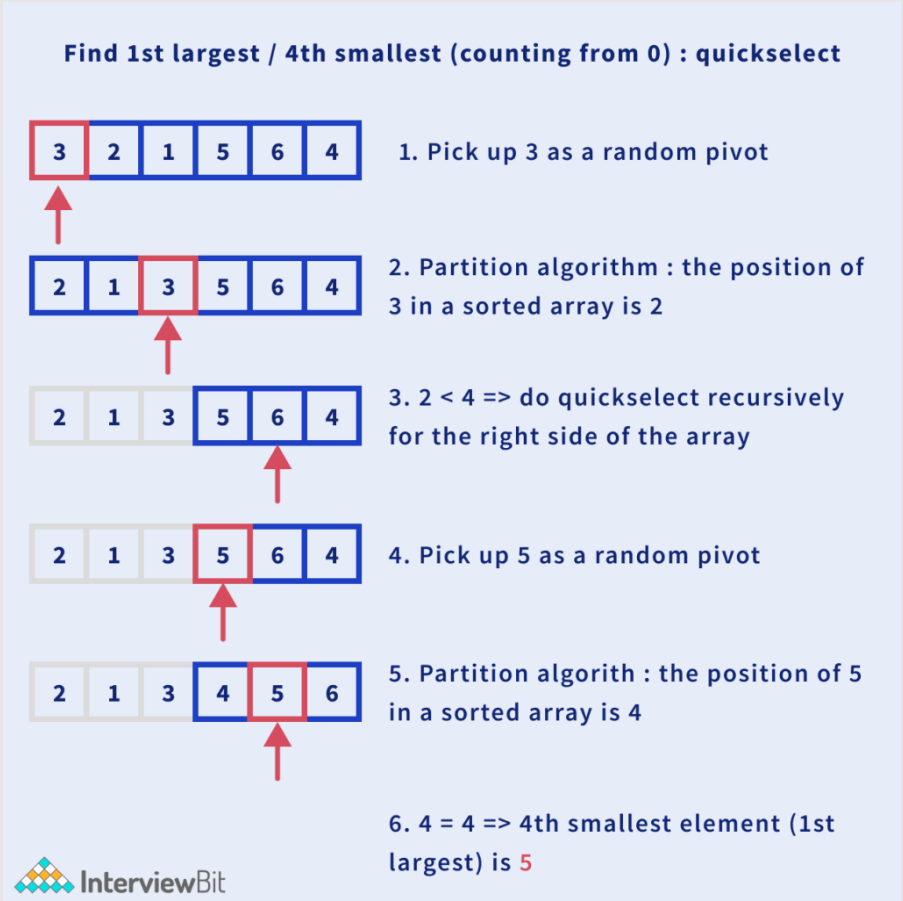
We have array A, we divide it to L = A[0:mid] and R = A[mid:n]

- Let’s solve task for two parts of array A - L and R.
- Let's solve the problem for the intersection of L and R.

k-th element


1 4 17 7 25 3 100
K = 2
Finding the kth smallest
and largest element
2nd largest
element
1 4 17 7 25 3 100
2nd smallest
element
K'th smallest element is : 3
K'th largest element is : 25
© w3resource.com

Algorithm


Find 1st largest / 4th smallest (counting from 0) : quickselect
3 2 1 5 6 4
1. Pick up 3 as a random pivot
2 1 3 5 6 4
2. Partition algorithm : the position of 3 in a sorted array is 2
2 1 3 5 6 4
3. 2 < 4 => do quickselect recursively for the right side of the array
2 1 3 5 6 4
4. Pick up 5 as a random pivot
2 1 3 4 5 6
5. Partition algorith : the position of 5 in a sorted array is 4
6. 4 = 4 => 4th smallest element (1st largest) is 5
InterviewBit

```
1.  def partition(a, l, r):
2.      pivot = a[r];
3.      pIndex = l;
4.
5.      for i in range(l, r):
6.          if (a[i] <= pivot):
7.              a[i], a[pIndex] = a[pIndex], a[i]
8.              pIndex += 1
9.      a[pIndex], a[r] = a[r], a[pIndex]
10.
11.     return pIndex;
12.
```

```
6.  int partition(vector<int> &a, int l, int r) {
7.      int pivot = a[r];
8.      int pIndex = l;
9.
10.     for (int i = l; i < r; i++) {
11.         if (a[i] <= pivot) {
12.             swap(a[i], a[pIndex]);
13.             pIndex++;
14.         }
15.     }
16.     swap(a[pIndex], a[r]);
17.
18.     return pIndex;
19. }
20.
```

```cpp
int quicksort(vector<int> &a, int start, int end, int k) {
    if (start >= end) {
        return a[start];
    }

    int pivot = partition(a, start, end);
    if (pivot - start > k) {
        return quicksort(a, start, pivot - 1, k);
    }
    if (pivot - start == k) {
        return a[pivot];
    }
    return quicksort(a, pivot + 1, end, k - (pivot - start));
}

int main() {
    vector<int> a = {11, 2, 3, 22, 1, 33, 5};

    for (int i = 0; i < a.size(); i++) {
        cout << quicksort(a, 0, a.size() - 1, i) << " ";
    }
    return 0;
}
```

#stdin #stdout 0.01s 5476KB

comments (0)

stdin

copy

```
Standard input is empty
```

stdout

copy

```
1 2 3 5 11 22 33
```

```python
def quicksort(a, start, end, k):
    if (start >= end):
        return

    pivot = partition(a, start, end)
    if pivot - start > k:
        return quicksort(a, start, pivot - 1, k)
    if pivot - start == k:
        return a[pivot]
    return quicksort(a, pivot + 1, end, k - (pivot - start))

a = list(map(int, input().split()))

for i in range(len(a)):
    print(quicksort(a, 0, len(a) - 1, i))

print(a)
```

#stdin #stdout 0.04s 9476KB

comments (0)

stdin

copy

```
2 33 1 3 4 5 12
```

stdout

copy

```
1
2
3
4
5
12
33
```

Asymptotic

- It can be strictly proved that such an algorithm runs for O(n)
- We can non-strictly through 2 * (n + 3/4n + 9/16n + ...) using reasoning as in quick sorting

sum (3/4)^j, j=0..infinity

NATURAL LANGUAGE | MATH INPUT

Infinite sum

$$\sum_{j=0}^{\infty}\left(\frac{3}{4}\right)^{j}=4$$

All codes

Merge sort(python) - https://ideone.com/8ZSRn1

merge sort(c++) - https://ideone.com/dJ6ZrY

Insertion sort(python) - https://ideone.com/Kl61lD

Selection sort(python) - https://ideone.com/uJbZcG

selection sort(c++) - https://ideone.com/QKJpd8

Insertion sort(c++) - https://ideone.com/cgJLs0

log(python) - https://ideone.com/Nx04xN

bubble sort(python) - https://ideone.com/O23FtD

bubble sort(c++) - https://ideone.com/Id6hx7

quick sort(c++) - https://ideone.com/2yGyYa

quick sort(python) - https://ideone.com/6Gqoxi

count sort(python) - https://ideone.com/5KOQH6

inversion count(python) - https://ideone.com/nyVFz4

inversion count(c++) - https://ideone.com/9alzlE

k-th element(python) - https://ideone.com/EXOKR9

k-th element(c++) - https://ideone.com/4jhbPl

That's All Folks!