Data structures and algorithms

Tutorial 5

Amr Keleg

Faculty of Engineering, Ain Shams University

March 19, 2020

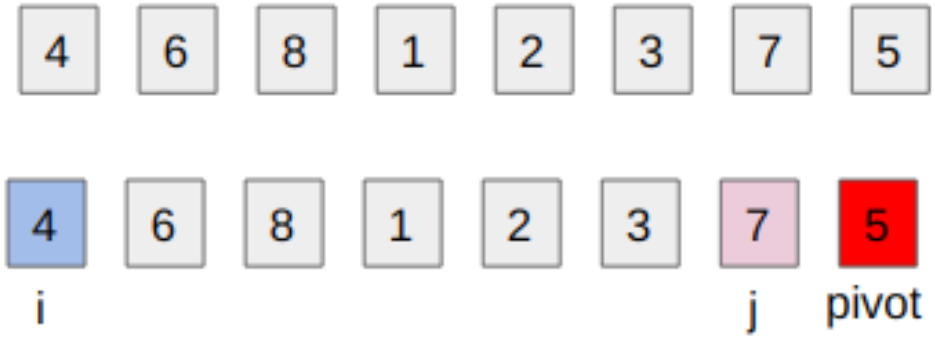
Contact: amr_mohamed@live.com

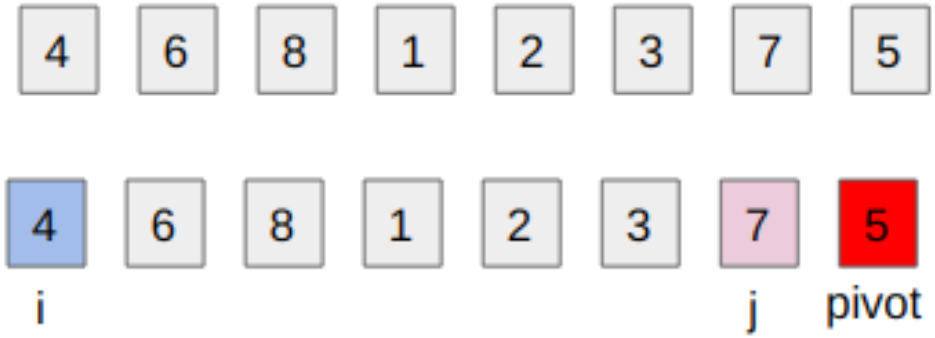
Outline

Outline

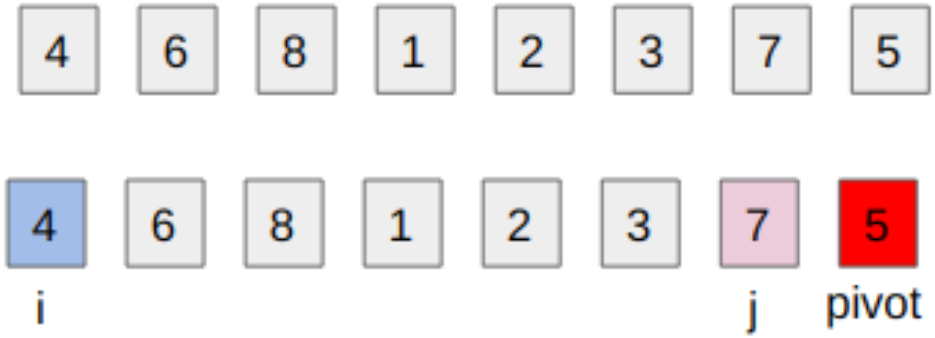
```
void quick_sort(int arr[], int l, int r);
```


4
6
8
1
2
3
7
5

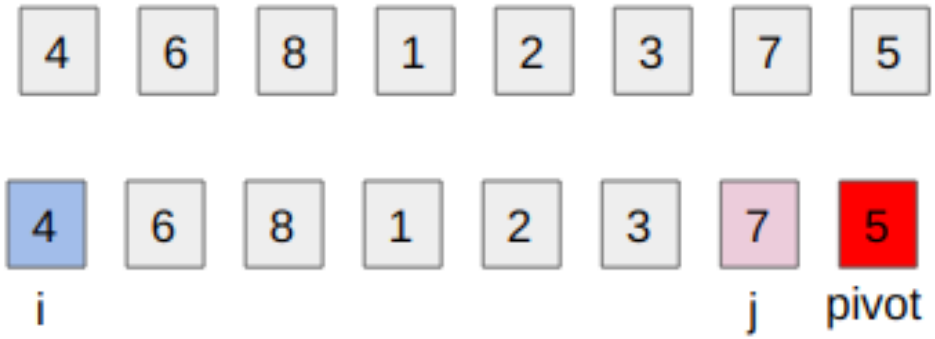

4
6
8
1
2
3
7
5
4
6
8
1
2
3
7
5
i
j
pivot



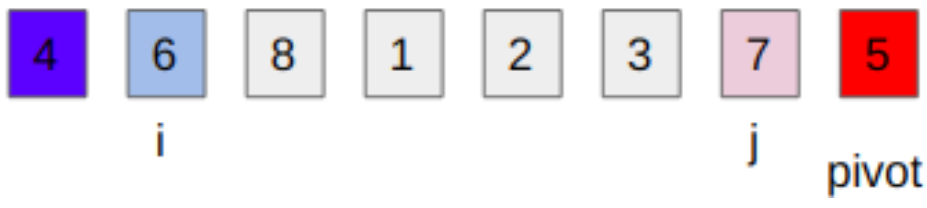
Is arr[i] $\leq$ pivot?

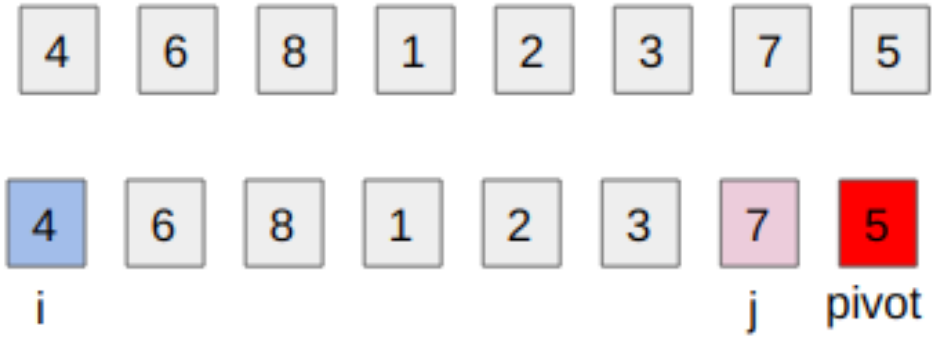


Is arr[i] $\leq$ pivot?Yes, i++

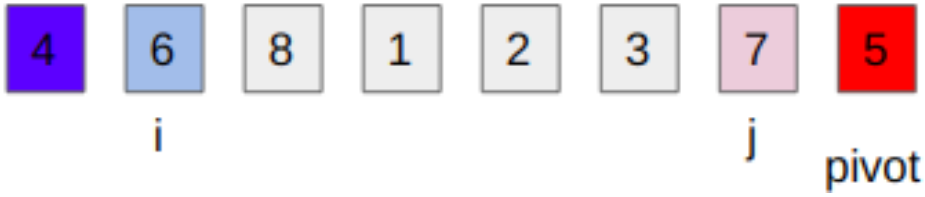


Is arr[i] $\leq$ pivot?Yes, i++

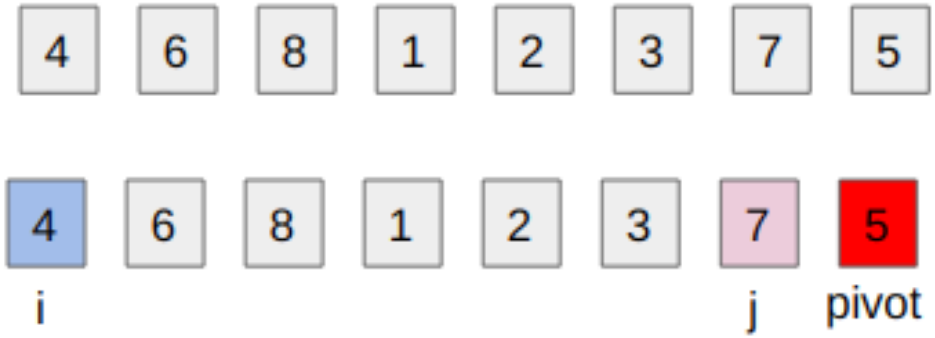




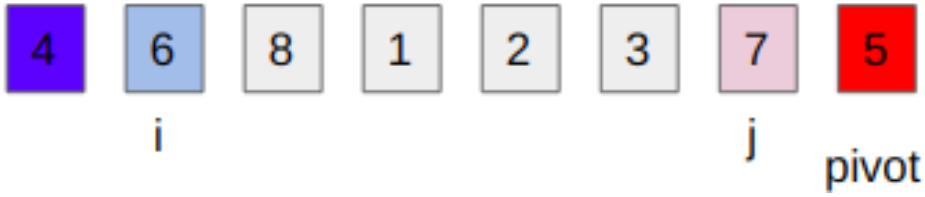
Is arr[i] $\leq$ pivot?Yes, i++



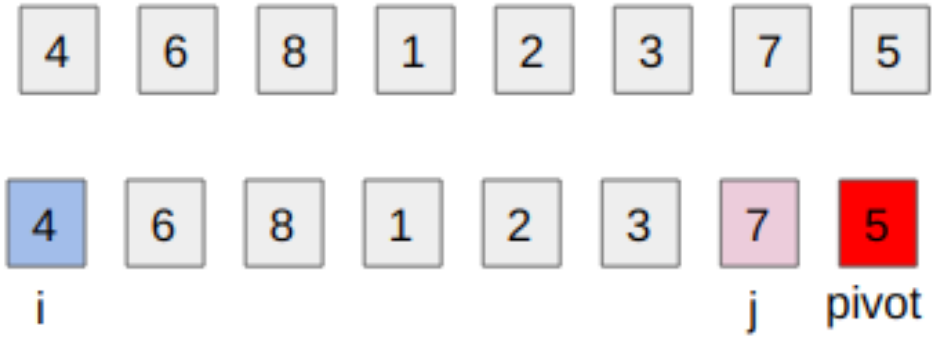
Is arr[i] $\leq$ pivot?



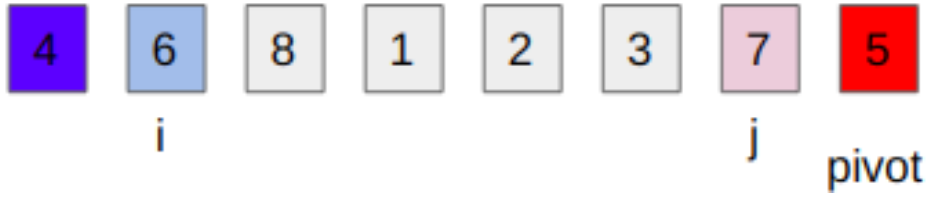
Is arr[i] $\leq$ pivot?Yes, i++



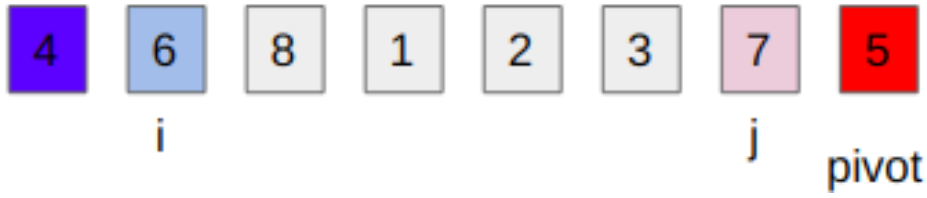
Is arr[i] $\leq$ pivot?No, Fix i

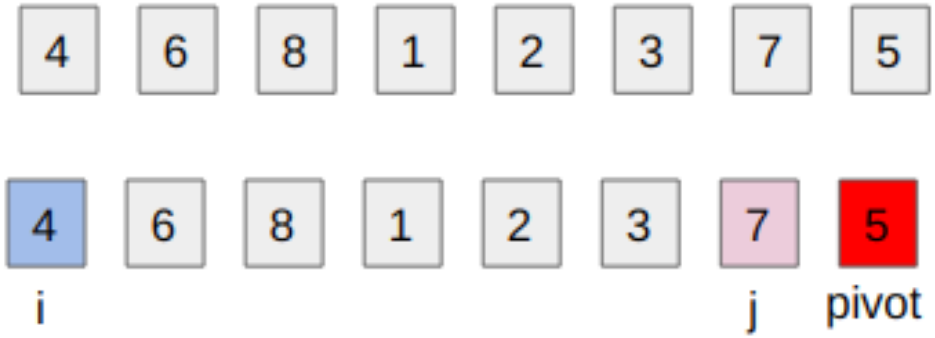


Is arr[i] $\leq$ pivot?Yes, i++

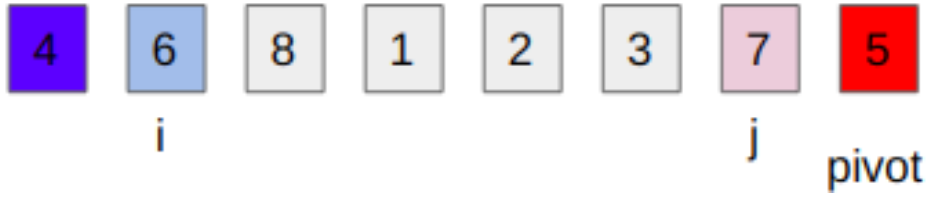


Is arr[i] $\leq$ pivot?No, Fix i

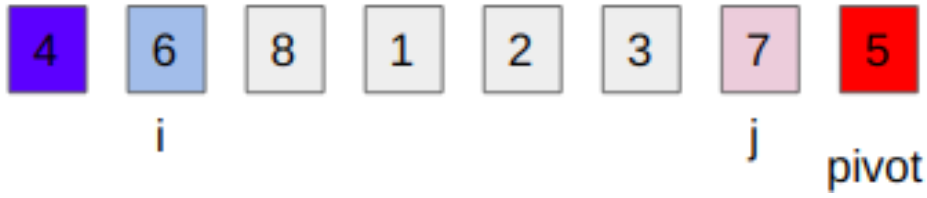




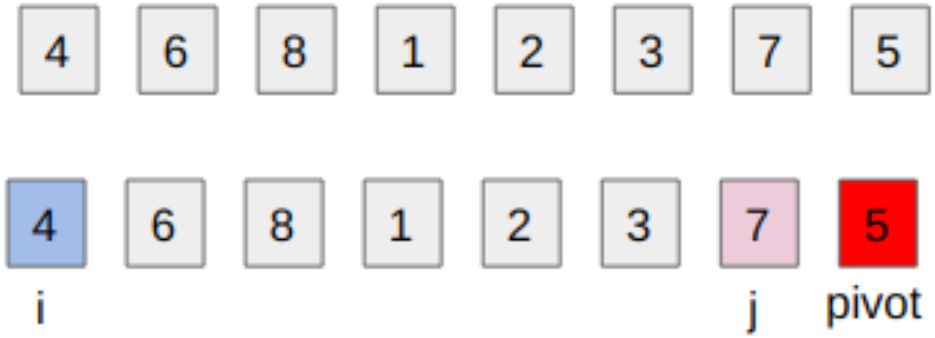
Is arr[i] $\leq$ pivot?Yes, i++



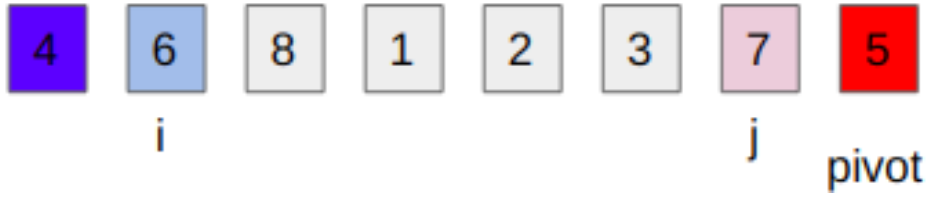
Is arr[i] $\leq$ pivot?No, Fix i



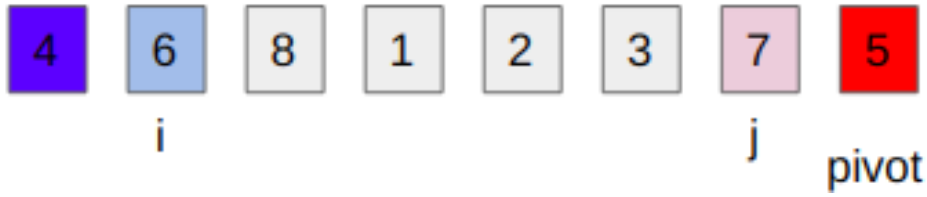
Is arr[j] >pivot?



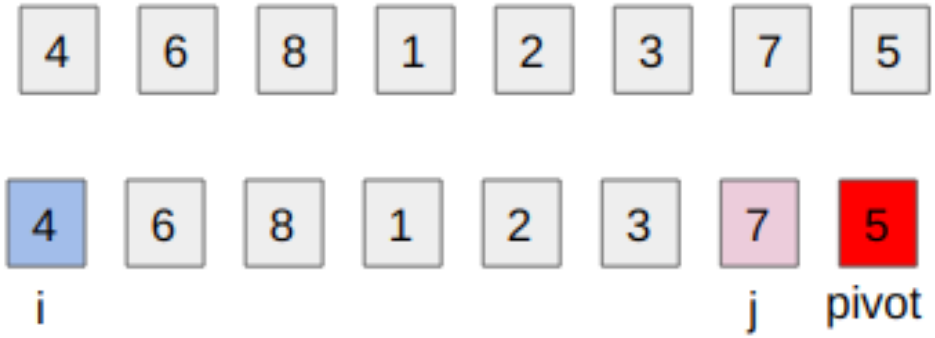
Is arr[i] $\leq$ pivot?Yes, i++



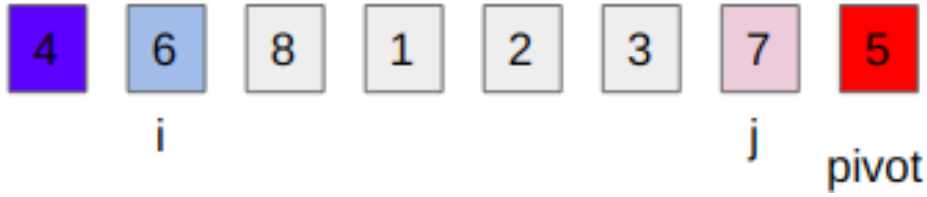
Is arr[i] $\leq$ pivot?No, Fix i



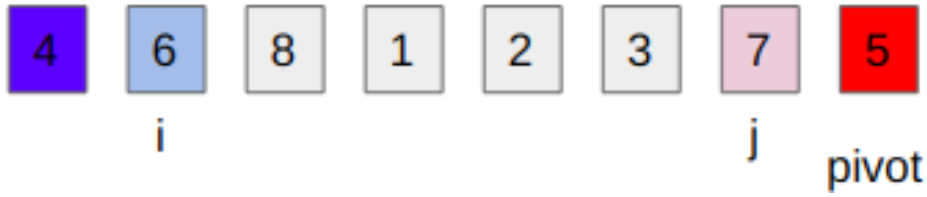
Is arr[j] >pivot?Yes, j - -



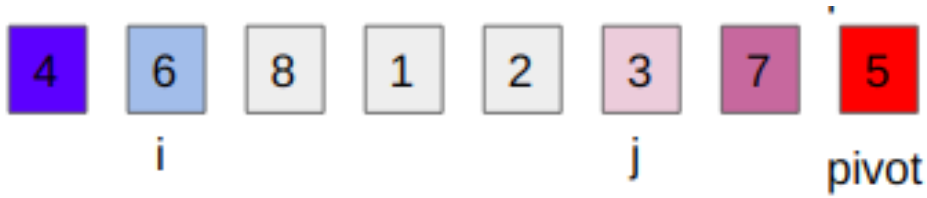
Is arr[i] $\leq$ pivot?Yes, i++

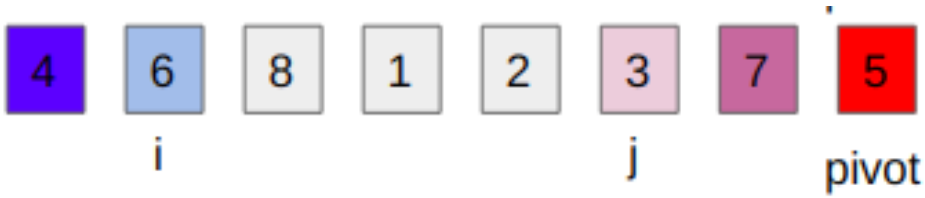


Is arr[i] $\leq$ pivot?No, Fix i

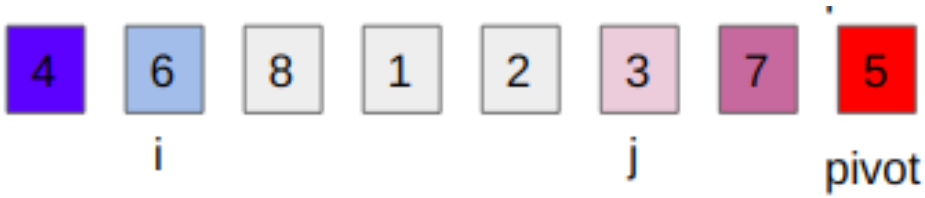


Is arr[j] >pivot?Yes, j - -

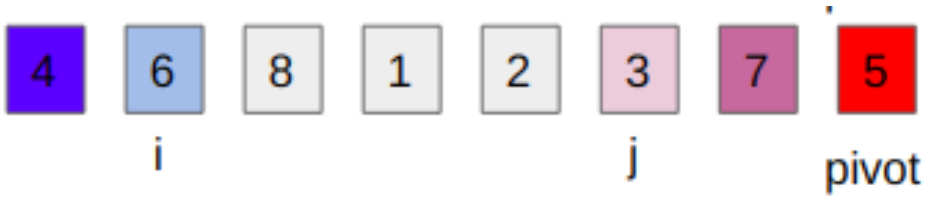

4
6
8
1
2
3
7
5
i
j
pivot



Is arr[j] >pivot?

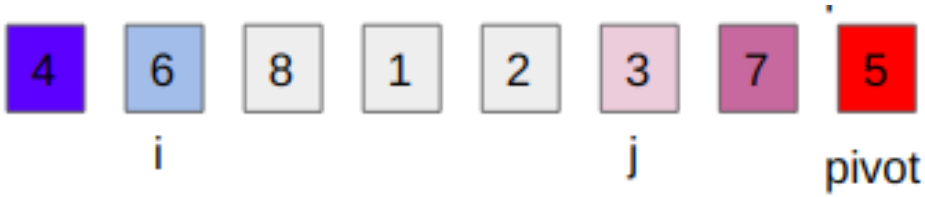


Is arr[j] >pivot?No, Fix j



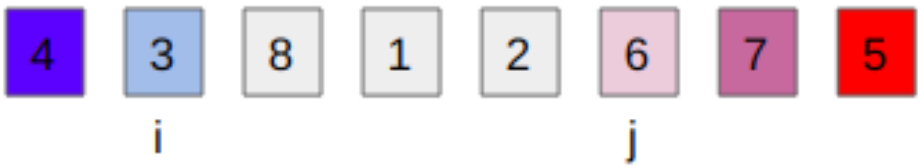
Is arr[j] >pivot?No, Fix j

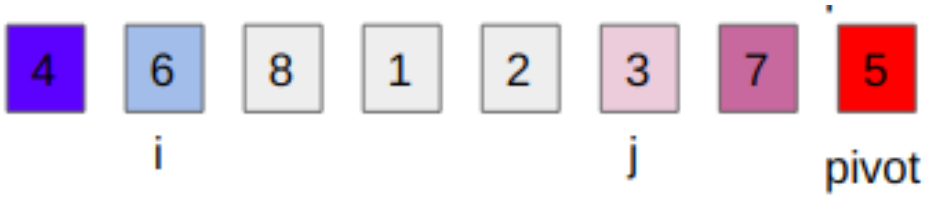
Now, swap the elements arr[i], arr[j]



Is arr[j] >pivot?No, Fix j

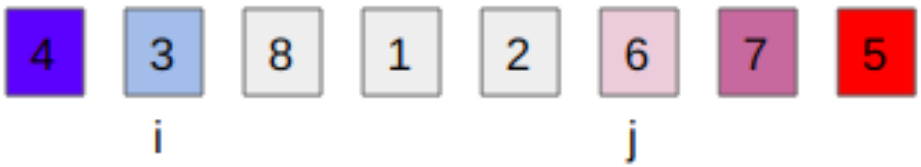
Now, swap the elements arr[i], arr[j]



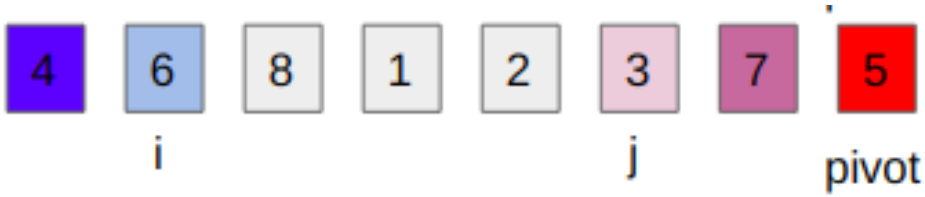


Is arr[j] >pivot?No, Fix j

Now, swap the elements arr[i], arr[j]

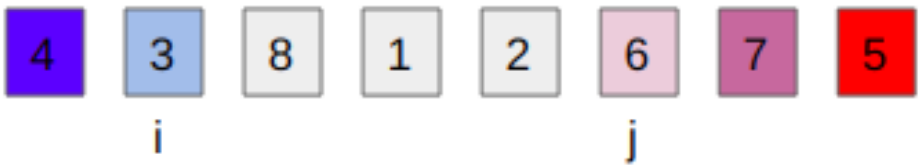


And REPEAT, Is arr[i] $\leq$ pivot?

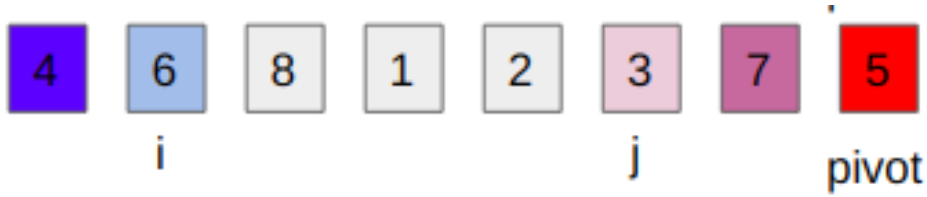


Is arr[j] >pivot?No, Fix j

Now, swap the elements arr[i], arr[j]

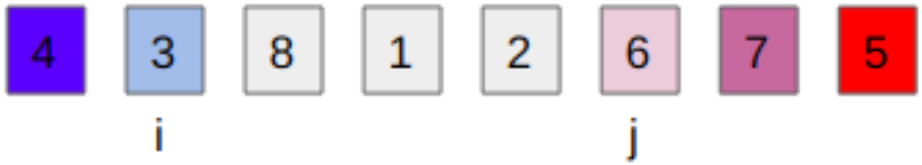


And REPEAT, Is arr[i] $\leq$ pivot?Yes, i++

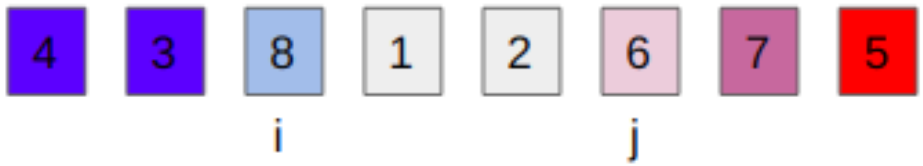


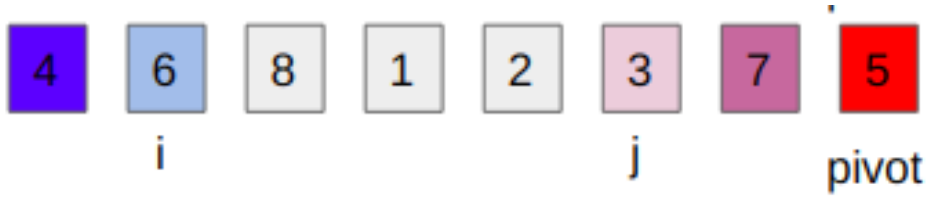
Is arr[j] >pivot?No, Fix j

Now, swap the elements arr[i], arr[j]



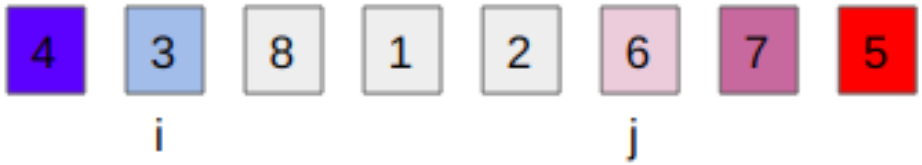
And REPEAT, Is arr[i] $\leq$ pivot?Yes, i++



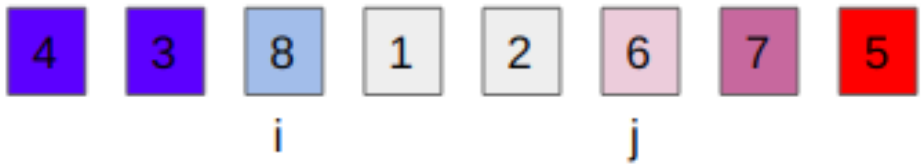


Is arr[j] >pivot?No, Fix j

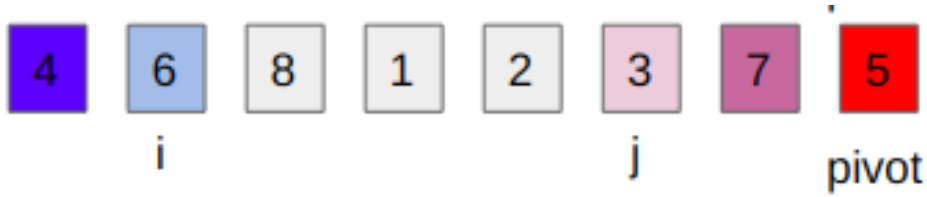
Now, swap the elements arr[i], arr[j]



And REPEAT, Is arr[i] $\leq$ pivot?Yes, i++

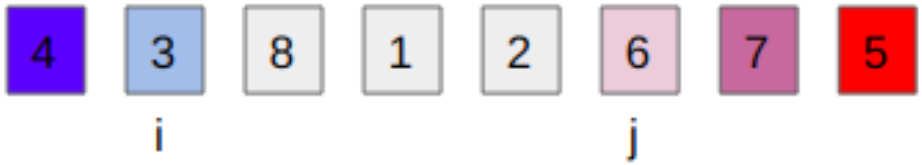


Is arr[i] $\leq$ pivot?

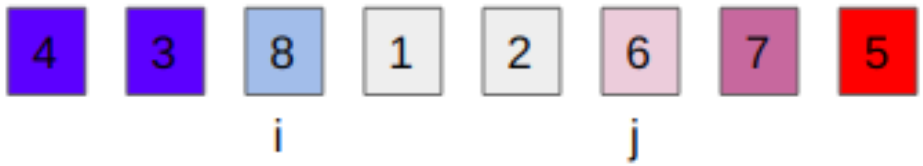


Is arr[j] >pivot?No, Fix j

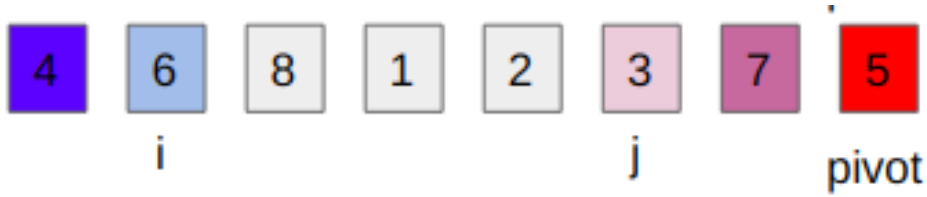
Now, swap the elements arr[i], arr[j]



And REPEAT, Is arr[i] $\leq$ pivot?Yes, i++

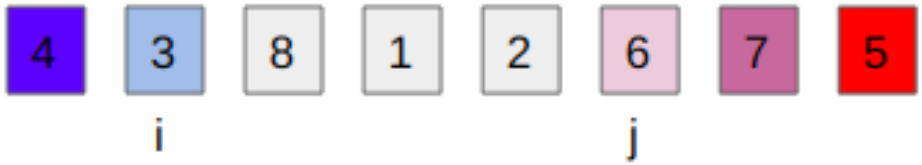


Is arr[i] $\leq$ pivot?No, Fix i

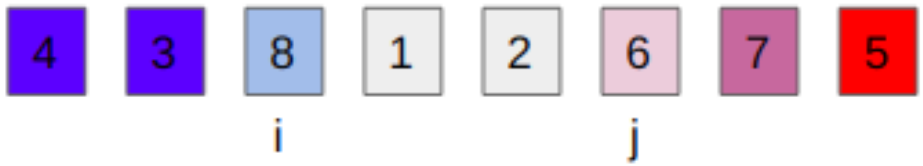


Is arr[j] >pivot?No, Fix j

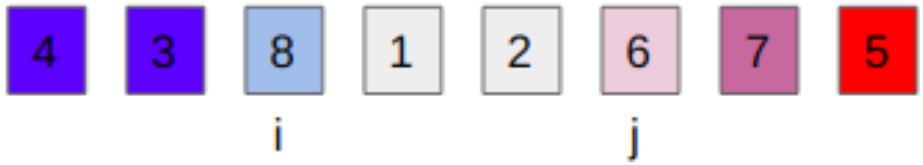
Now, swap the elements arr[i], arr[j]

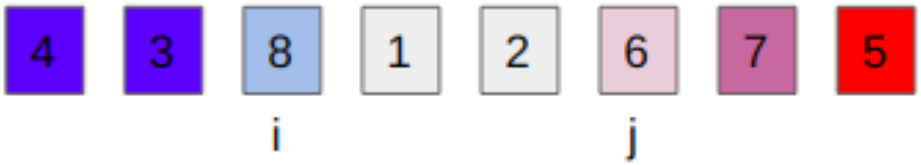


And REPEAT, Is arr[i] $\leq$ pivot?Yes, i++

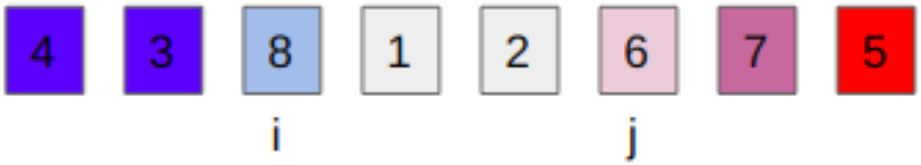


Is arr[i] $\leq$ pivot?No, Fix i

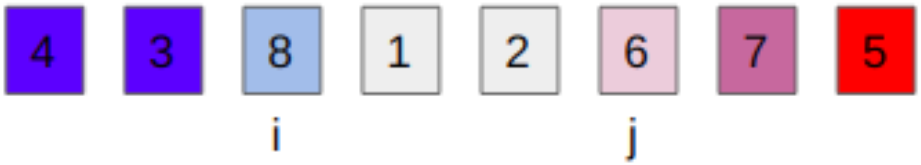

4
3
8
1
2
6
7
5
i
j



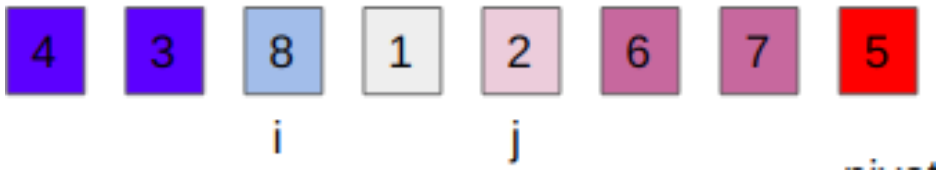
Is arr[j] >pivot?

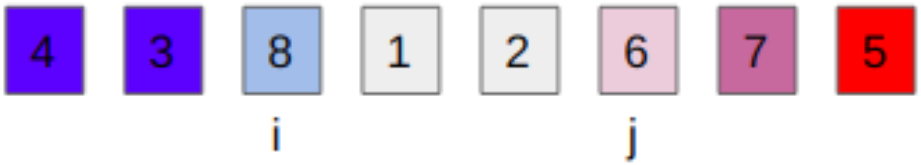


Is arr[j] >pivot?Yes, j - -

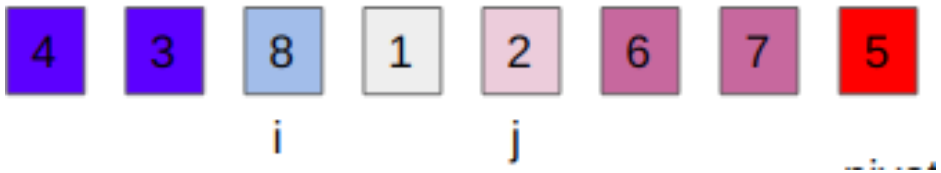


Is arr[j] >pivot?Yes, j - -

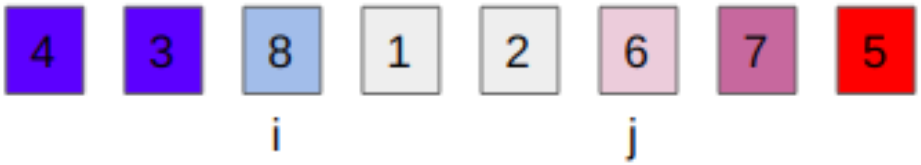




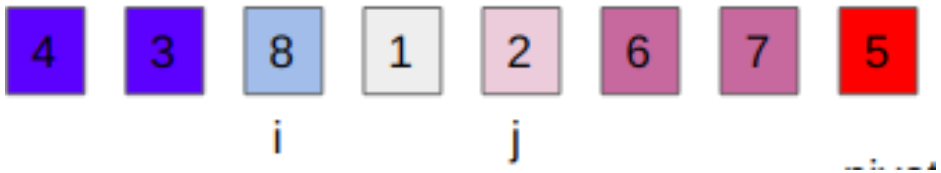
Is arr[j] >pivot?Yes, j - -



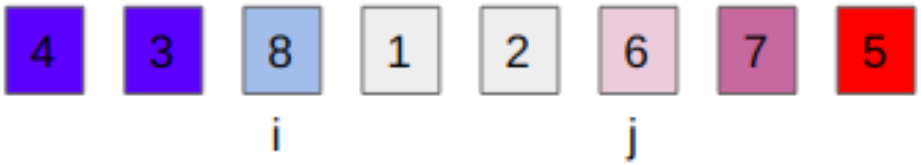
Is arr[j] >pivot?



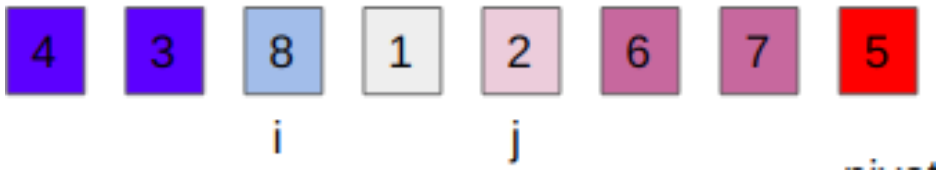
Is arr[j] >pivot?Yes, j - -



Is arr[j] >pivot?No, fix j

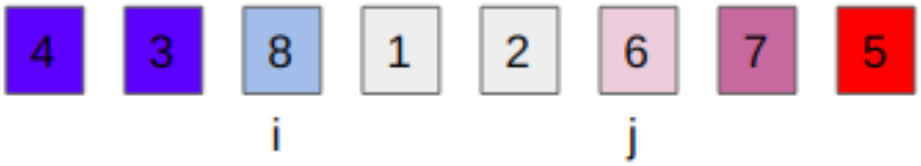


Is arr[j] >pivot?Yes, j - -

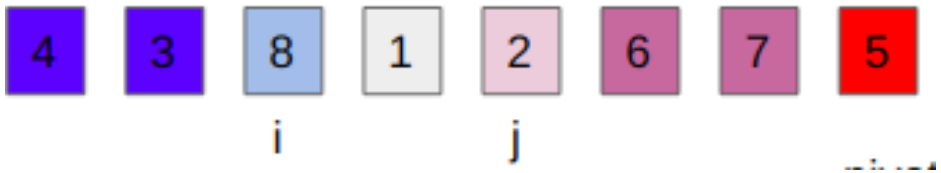


Is arr[j] >pivot?No, fix j

Now, swap the elements arr[i], arr[j]

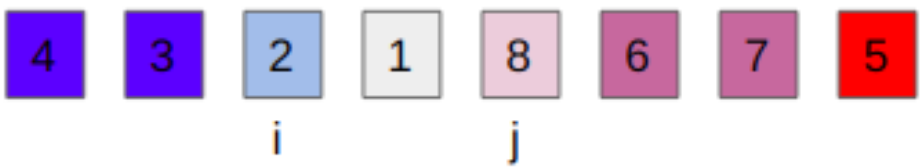


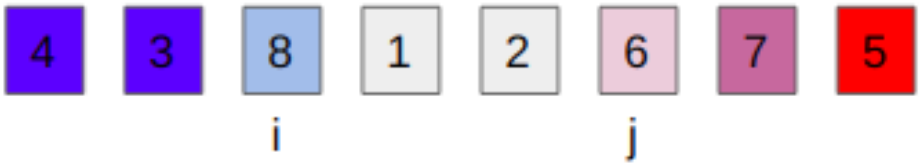
Is arr[j] >pivot?Yes, j - -



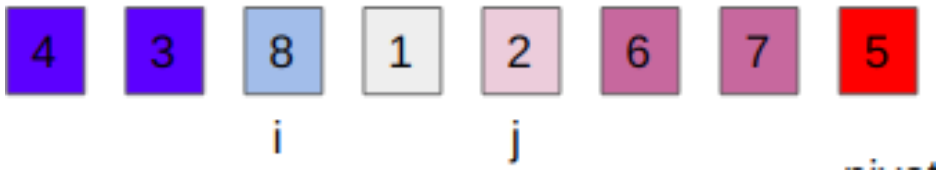
Is arr[j] >pivot?No, fix j

Now, swap the elements arr[i], arr[j]



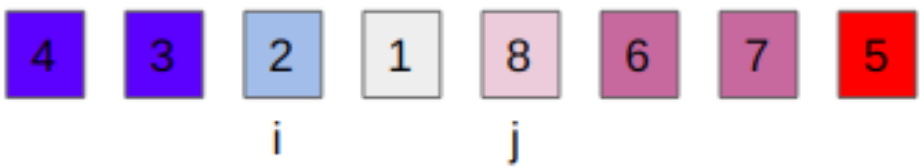


Is arr[j] >pivot?Yes, j - -

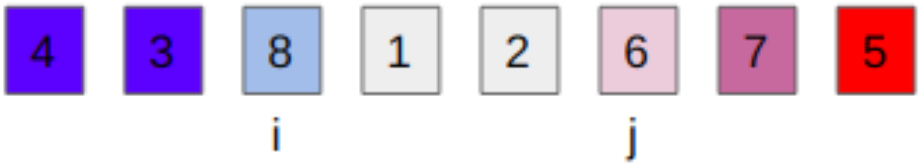


Is arr[j] >pivot?No, fix j

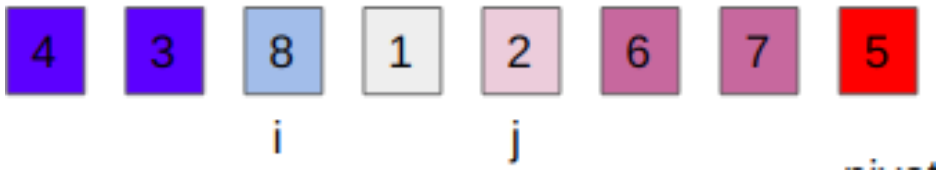
Now, swap the elements arr[i], arr[j]



And REPEAT, Is arr[i] $\leq$ pivot?

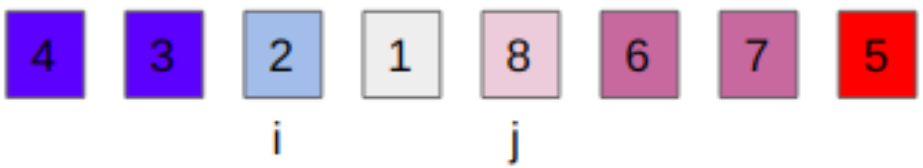


Is arr[j] >pivot?Yes, j - -

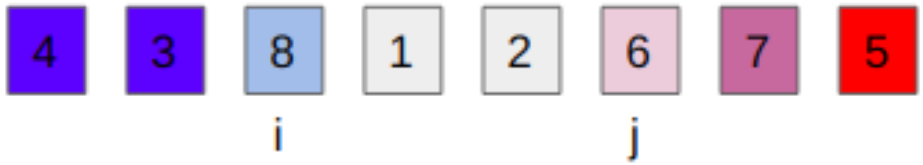


Is arr[j] >pivot?No, fix j

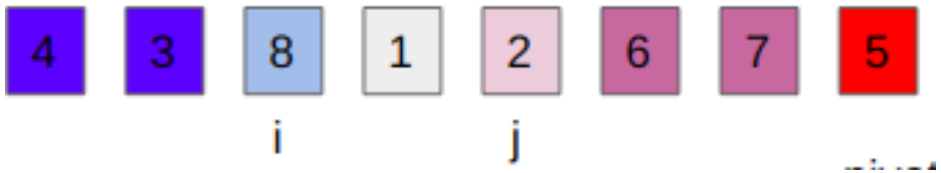
Now, swap the elements arr[i], arr[j]



And REPEAT, Is arr[i] $\leq$ pivot?Yes, i++

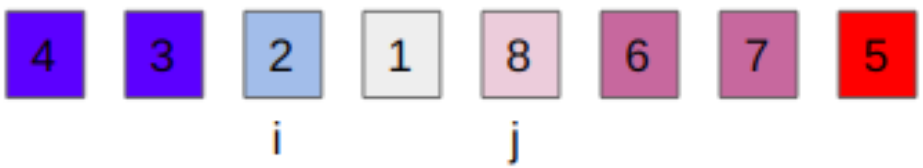


Is arr[j] >pivot?Yes, j - -

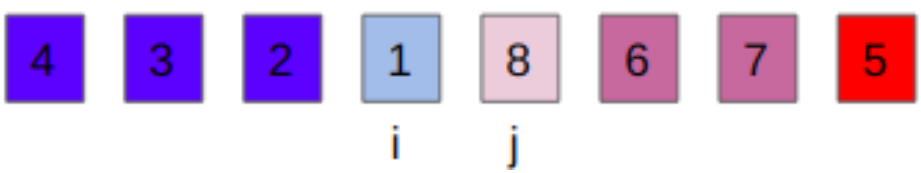


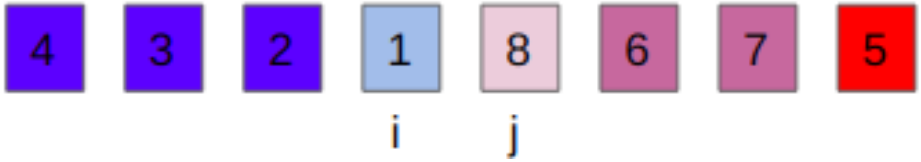
Is arr[j] >pivot?No, fix j

Now, swap the elements arr[i], arr[j]

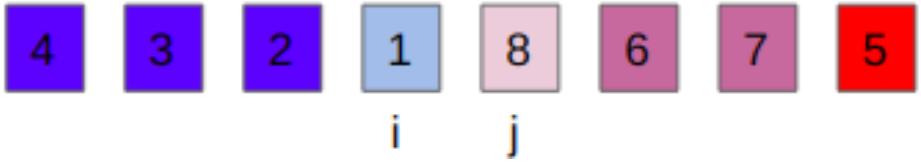


And REPEAT, Is arr[i] $\leq$ pivot?Yes, i++

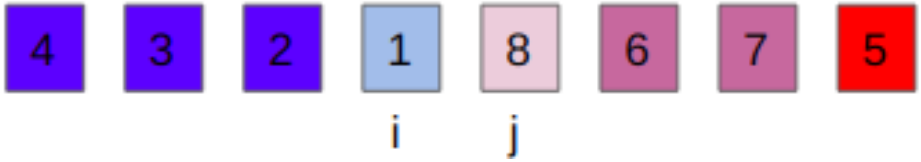




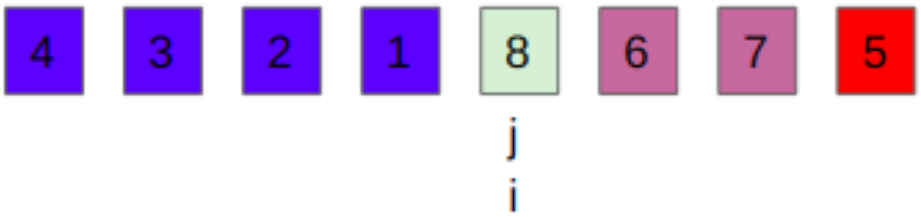
Is arr[i] $\leq$ pivot?

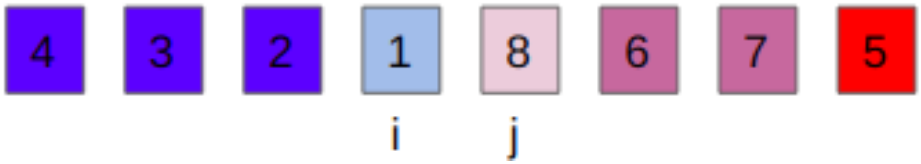


Is arr[i] $\leq$ pivot?Yes, i++

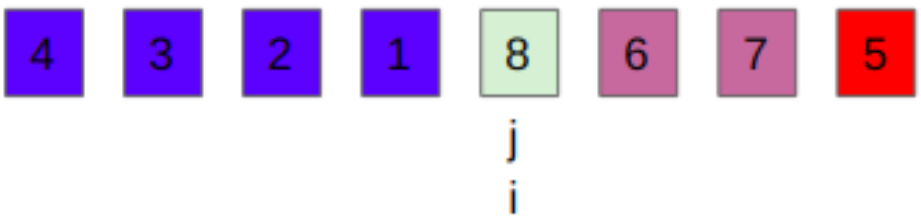


Is arr[i] $\leq$ pivot?Yes, i++

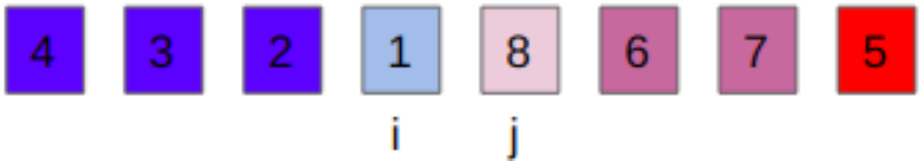




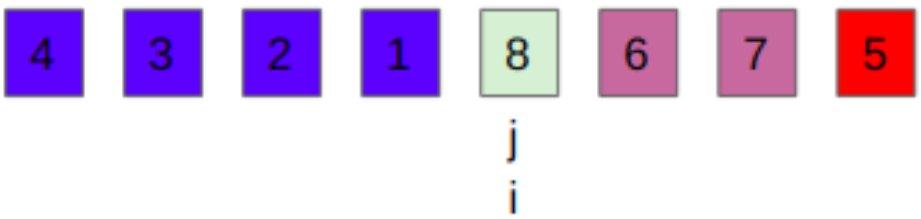
Is arr[i] $\leq$ pivot?Yes, i++



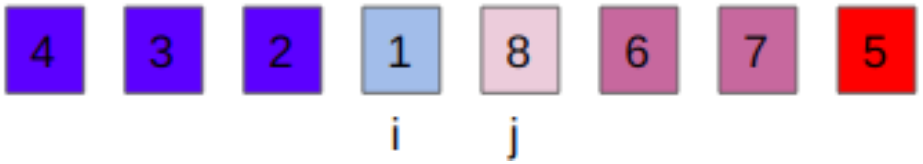
Is arr[i] $\leq$ pivot?



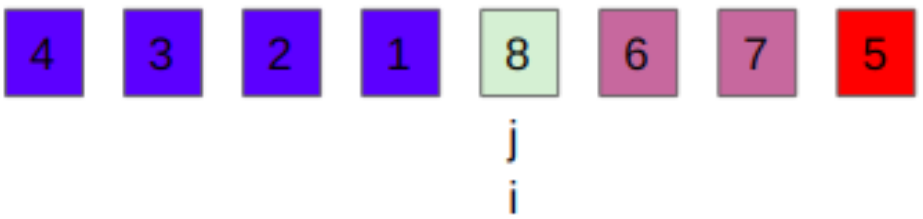
Is arr[i] $\leq$ pivot?Yes, i++



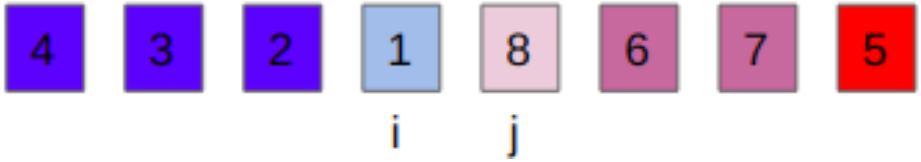
Is arr[i] $\leq$ pivot?No, Fix i



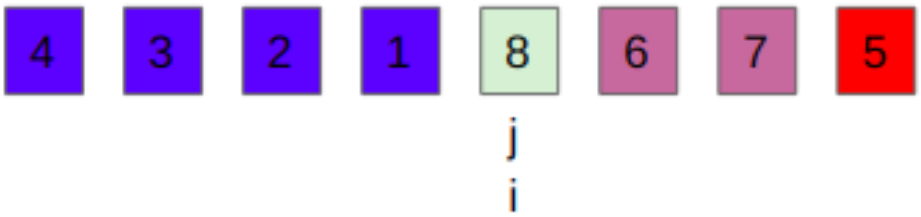
Is arr[i] $\leq$ pivot?Yes, i++



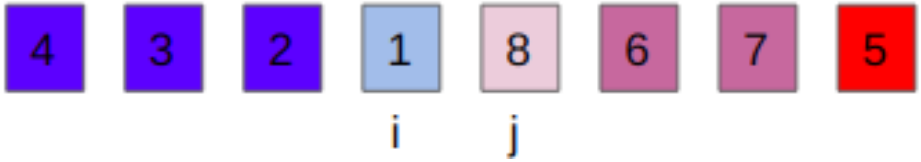
Is arr[i] $\leq$ pivot?No, Fix i
Is arr[j] $>$pivot?



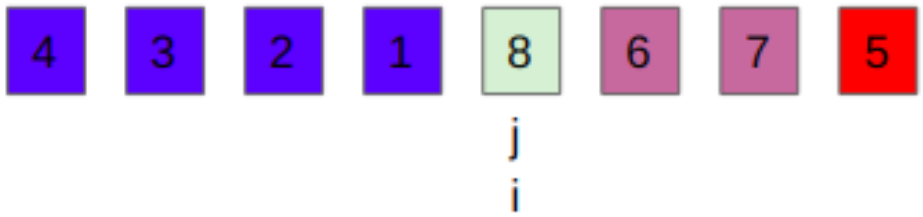
Is arr[i] $\leq$ pivot?Yes, i++



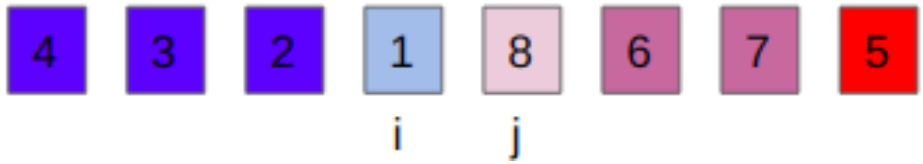
Is arr[i] $\leq$ pivot?No, Fix i
Is arr[j] $>$pivot?Yes, j - -



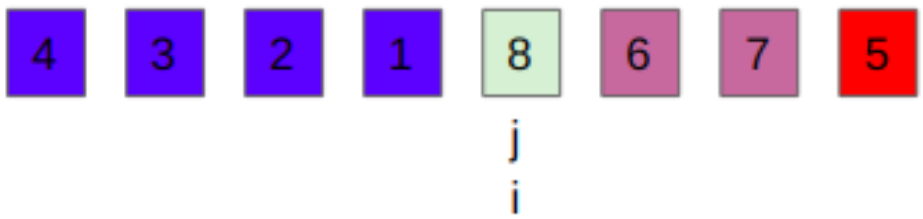
Is arr[i] $\leq$ pivot?Yes, i++



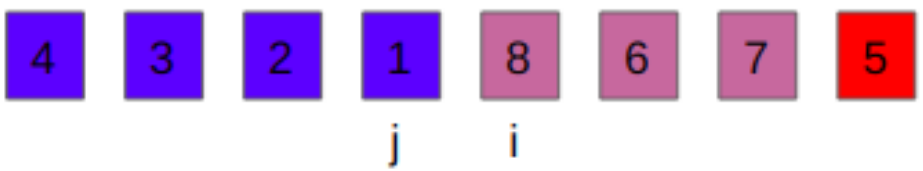
Is arr[i] $\leq$ pivot?No, Fix i
Is arr[j] $>$pivot?Yes, j - -



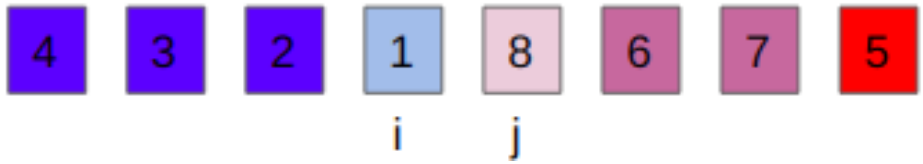
Is arr[i] $\leq$ pivot?Yes, i++



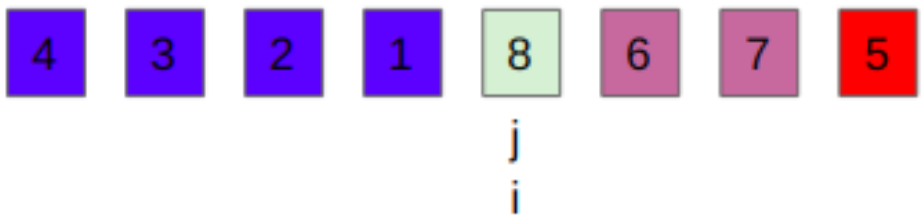
Is arr[i] $\leq$ pivot?No, Fix i
Is arr[j] $>$pivot?Yes, j - -



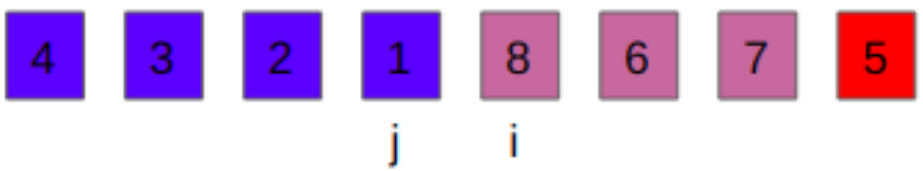
Is arr[j] $>$pivot?



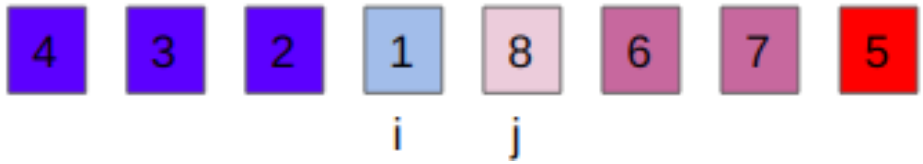
Is arr[i] $\leq$ pivot?Yes, i++



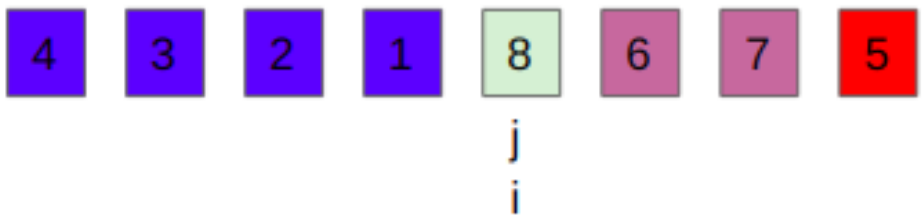
Is arr[i] $\leq$ pivot?No, Fix i
Is arr[j] $>$pivot?Yes, j - -



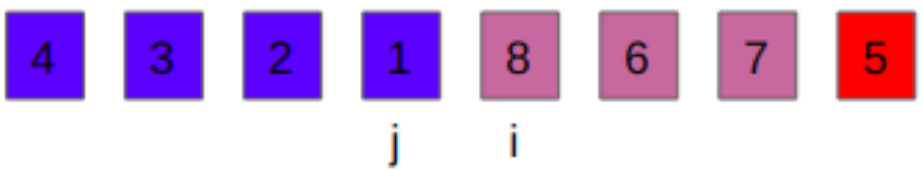
Is arr[j] $>$pivot?No, Fix j



Is arr[i] $\leq$ pivot?Yes, i++



Is arr[i] $\leq$ pivot?No, Fix i
Is arr[j] >pivot?Yes, j - -



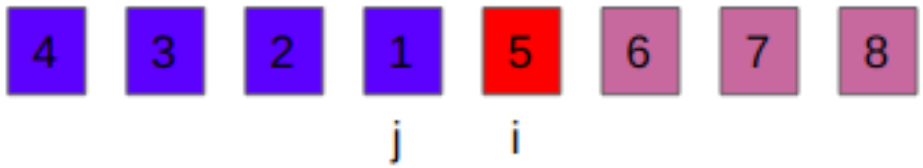
Is arr[j] >pivot?No, Fix j

Since j <i then we are done.

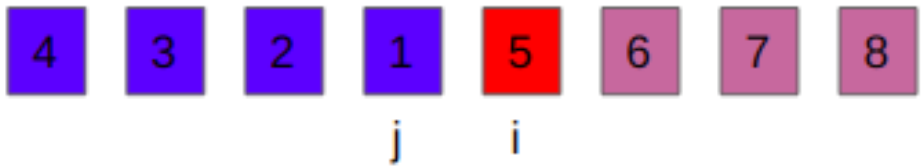

4
3
2
1
8
6
7
5
j
i

Where would you place the pivot?

Where would you place the pivot?swap arr[r], arr[i]



Where would you place the pivot?swap arr[r], arr[i]



Now, call the function recursively again on both sides.

```
void quick_sort(int arr[], int l, int r){
  if (l>=r) return;
  int pivot_val = arr[r];
  int i = l;
  int j = r-1;
  while(i<=j){
    while(arr[i] <= pivot_val && i<=r-1) i++;

    while(arr[j] > pivot_val && j>=l) j--;

    if (i<j)
      swap(arr[i], arr[j]);
  }
  // Move pivot element to the correct location (i)
  swap(arr[r], arr[i]);
  quick_sort(arr, l, j);
  quick_sort(arr, i+1, r);
}
```

Complexity of Quick Sort?

Complexity of Quick Sort?
Single call is O(n) (Pass through all the elements and assign each one to one of the two halves).

Complexity of Quick Sort?
Single call is O(n) (Pass through all the elements and assign each one to one of the two halves).
Worst Case?

Complexity of Quick Sort?
Single call is O(n) (Pass through all the elements and assign each one to one of the two halves).
Worst Case?
Array is sorted.

Complexity of Quick Sort?
Single call is O(n) (Pass through all the elements and assign each one to one of the two halves).
Worst Case?
Array is sorted.
Best Case?

Complexity of Quick Sort?
Single call is O(n) (Pass through all the elements and assign each one to one of the two halves).
Worst Case?
Array is sorted.
Best Case?
Pivot divides the array into two EQUAL parts

Complexity of Quick Sort?
Single call is O(n) (Pass through all the elements and assign each one to one of the two halves).
Worst Case?
Array is sorted.
Best Case?
Pivot divides the array into two EQUAL parts
Complexity? (hint: think of merge sort)

Complexity of Quick Sort?
Single call is O(n) (Pass through all the elements and assign each one to one of the two halves).
Worst Case?
Array is sorted.
Best Case?
Pivot divides the array into two EQUAL parts
Complexity? (hint: think of merge sort)
O(n * log(n))

Outline

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.

```
void selection_sort(int arr[], int arr_len)
{
  for(int i=0; i<arr_len - 1; i++){
    // find the minimum value in range [i, arr_len[
    int index_of_min = i;
    for(int j=i+1; j<arr_len; j++){
      if (arr[j] < arr[index_of_min])
        index_of_min = j;
    }
    // move the minimum value to index i
    swap(arr[index_of_min], arr[i]);
  }
}

int main(){
  int arr[] = {3, 1, 2};
  int arr_len = 3;
  selection_sort(arr, arr_len);
}
```

Outline

Q1: Modify the selection sort to be used for a linked list.

```
// Assume this is a method in the LinkedList class
class LinkedList{
private:
  Node * head;
  ...
public:
  ...
  void selection_sort()
};
```

```
void selection_sort(int arr[], int arr_len)
{
  for(int i=0; i<arr_len - 1; i++){
    // find the minimum value in range [i, arr_len[
    int index_of_min = i;
    for(int j=i+1; j<arr_len; j++){
      if (arr[j] < arr[index_of_min])
        index_of_min = j;
    }
    // move the minimum value to index i
    swap(arr[index_of_min], arr[i]);
  }
}
```

```
void selection_sort(){
  Node * cur_node = head;
  int list_length = this->length();
  for(int it=0; it<list_length-1; it++){
    Node * min_node = cur_node;
    Node * it_node = cur_node->next;
    while(it_node != nullptr){
      if (it_node-> data < min_node->data)
        min_node = it_node;
      it_node = it_node->next;
    }
    swap(min_node->data, cur_node->data);
    // Move the cur_node
    cur_node = cur_node->next;
  }
}
```

Q2: Bubble sort is based on swapping the consecutive arrays contents when they are not in order, is this sort method suitable for a linked list version than selection sort? Discuss

Q2: Bubble sort is based on swapping the consecutive arrays contents when they are not in order, is this sort method suitable for a linked list version than selection sort? Discuss
Answer: In Bubble sort, one needs to compare each node to the node next to it which is easy to do for Linked lists.

Q2: Bubble sort is based on swapping the consecutive arrays contents when they are not in order, is this sort method suitable for a linked list version than selection sort? Discuss
Answer: In Bubble sort, one needs to compare each node to the node next to it which is easy to do for Linked lists.
In Selection sort, we need to keep track of the position in which the minimum value will be placed at the end of each iteration.
But, in the end, they both have the same worst case complexity $O(n^2)$.

```
void bubble_sort(){
  if(empty())
    return;
  for(int it=0; it<length() - 1; it++){
    Node * pre_last_node = head;
    Node * last_node = head->next;
    while(last_node != nullptr){
      if (pre_last_node->data > last_node->data){
        swap(pre_last_node->data, last_node->data);
      }
      pre_last_node = last_node;
      last_node = last_node->next;
    }
  }
}
```

Outline

Outline

```
class Queue{
  public:
    Queue();
    int front();
    int back();
    void push(int v); // enqueue
    void pop(); //dequeue
};
```

Remember that a Queue is a FIFO (First In First Out) data structure.

How to implement a queue?

- A linked list
- An array of fixed size
- A dynamically allocated array
- A circular array

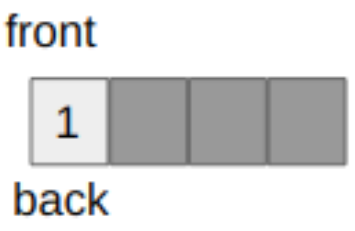
Let's simulate these operations for a static array implementation:

Let's simulate these operations for a static array implementation:

push 1

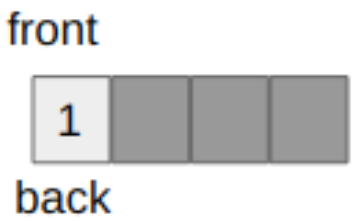
Let's simulate these operations for a static array implementation:

push 1



Let's simulate these operations for a static array implementation:

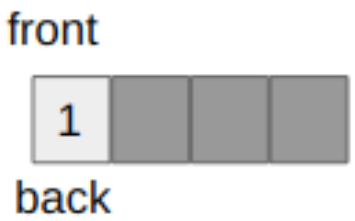
push 1



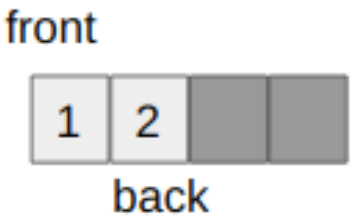
push 2

Let's simulate these operations for a static array implementation:

push 1

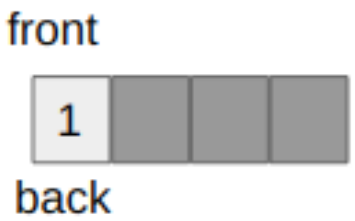


push 2

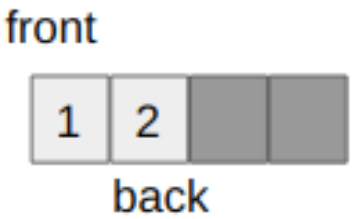


Let's simulate these operations for a static array implementation:

push 1

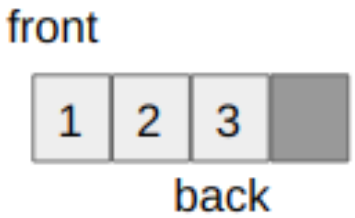


push 2

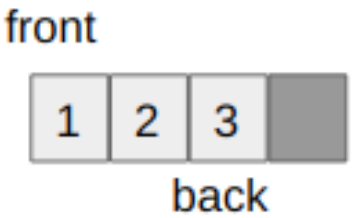


push 3

push 3

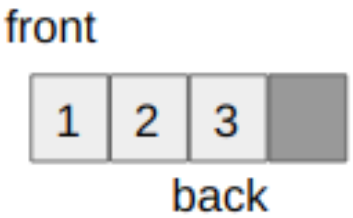


push 3

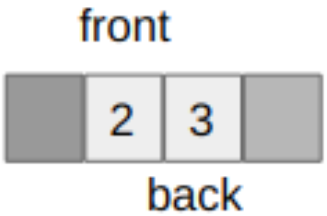


pop

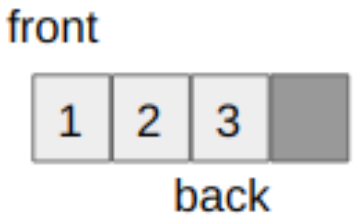
push 3



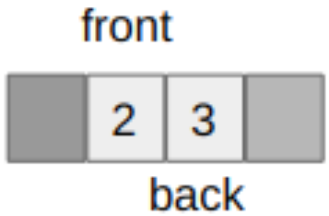
pop



push 3

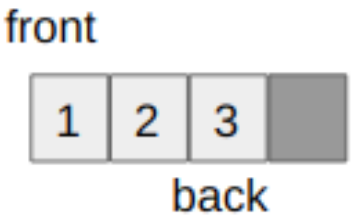


pop

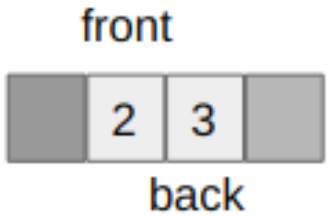


push 4

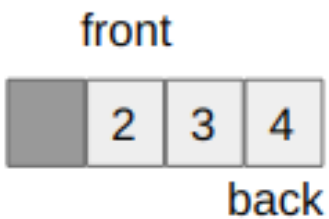
push 3



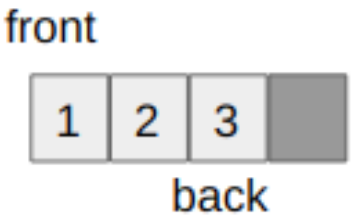
pop



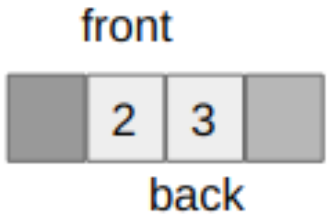
push 4



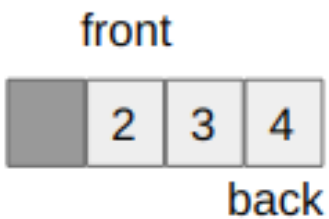
push 3

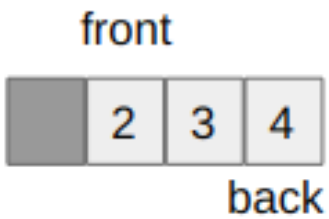


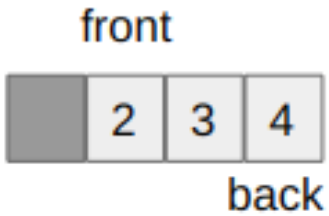
pop



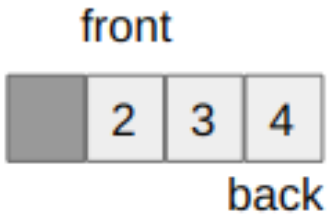
push 4




front
2
3
4
back

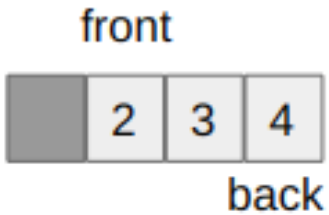


push 5, Problem!!



push 5, Problem!!

Solution:

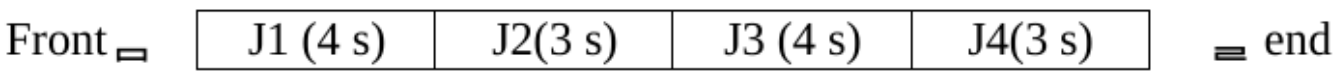


push 5, Problem!!

Solution:
back = (back + 1) % arr_size
front = (front + 1) % arr_size

Outline

- Using a queue of jobs where each job has an estimated service time, Develop a method to compute the required service time of all jobs in the queue and the average waiting time per job.
- Try not to destroy the queue after finding the sum of the jobs.
- Apply your method on the following queue of four jobs.



```
void job_statisitics(queue q){
  int q_size = q.size();
  int total_service_time = 0;
  int total_waiting_time = 0;
  for (int i=0; i<q_size; i++){
    int cur_time = q.front();
    q.pop();
    total_waiting_time += total_service_time;
    total_service_time += cur_time;
    q.push(cur_time);
  }

  // The Total time for this example is 14
  cout<< "Total time "<<total_service_time <<"\n";

  // The average waiting time for this example is 5.5
  cout<< "Average waiting time "
    <<1.0 * total_waiting_time / q_size<<"\n";
}
```

Feedback form:
Amr: `https://forms.gle/6q2XhFJ2FowQ7YqA9`

Discussion document:
`https://docs.google.com/document/d/1258ERLzJRreQN8VJbe5lFsvce6IWxLqAWpKFrB1K8_4/edit?usp=sharing`