Searching & Sorting

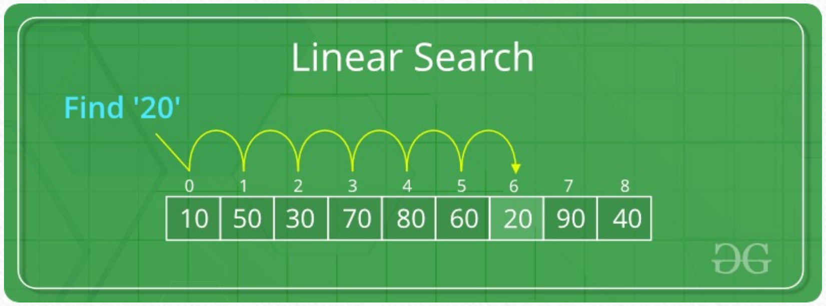
KNTU

Fall 2023

M.Abdolali

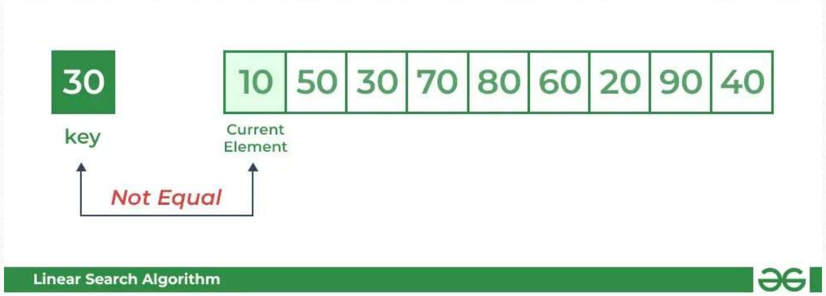
(1) Linear search

- Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored.
- the list or array is traversed sequentially and every element is checked

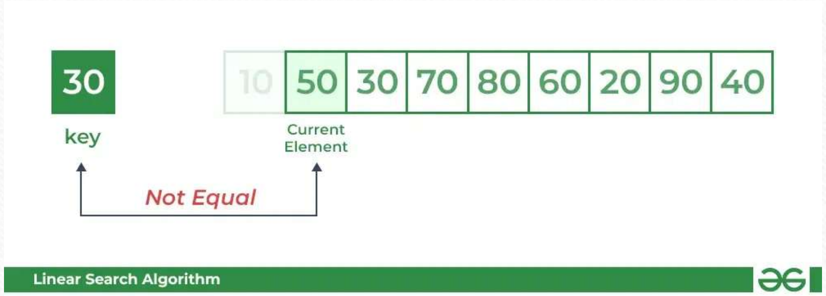


Linear search
-Analysis-

Step 1

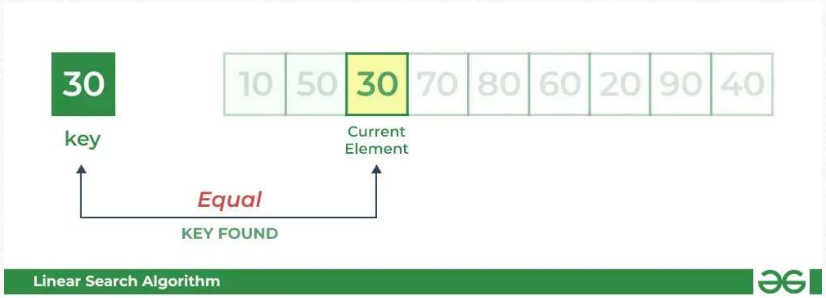


Step 2



Linear search
-Analysis-

Step 3



Analysis

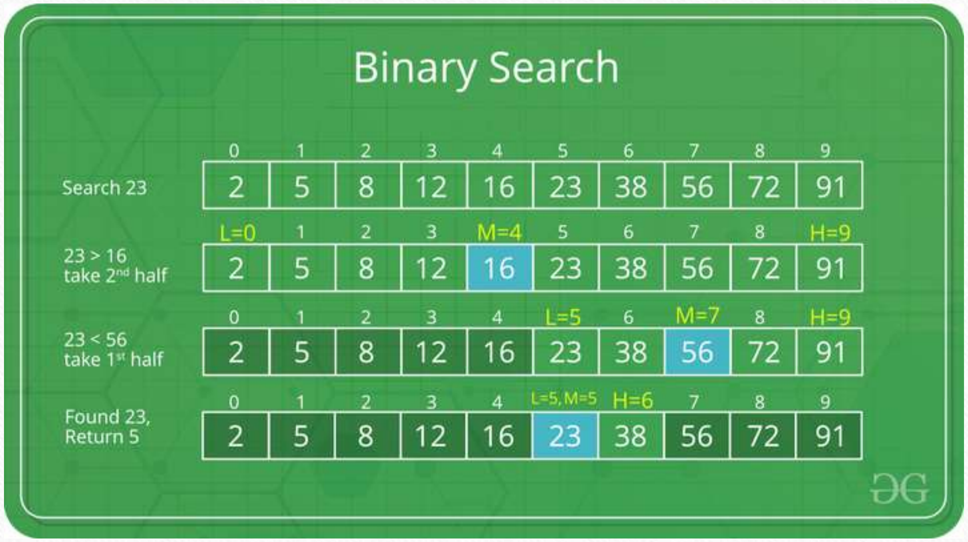
- Average time complexity: O(N)
- Auxiliary Space: O(1)

Linear search
-code-

```
arr = [2, 3, 4, 10, 40]
key = 3
index = -1
for i in range(0, len(arr)):
    if (arr[i] == key):
        index = i
        break
if index >=0:
    print(f"found key: {key}, at the index: {index}")
else:
    print("Did not find the key")
```

(2) Binary search

- specifically designed for searching in ***sorted data***
- repeatedly dividing the search interval in half

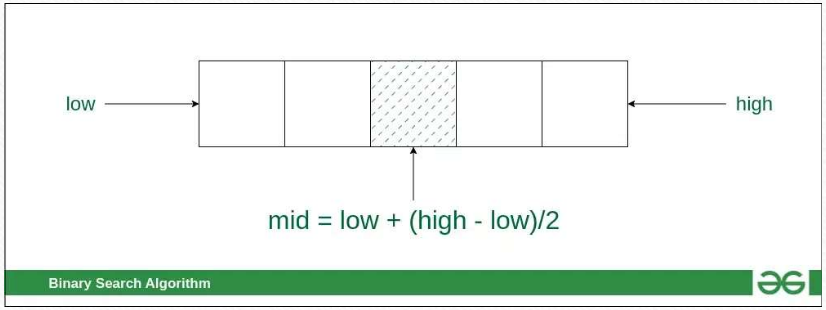


Binary search

Main steps

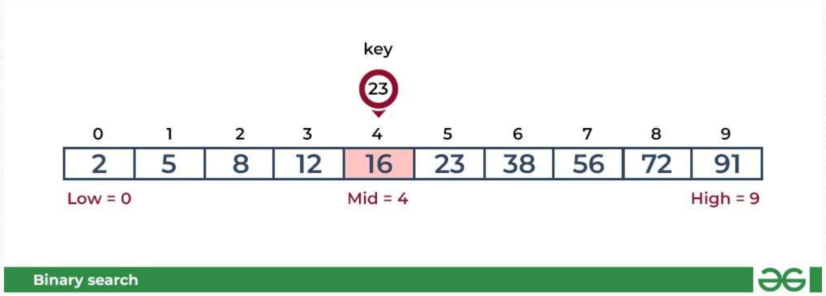
- 1. *Divide the search space into two halves by finding the middle index "mid"*
- *2. Compare the middle element of the search space with the key.*
- *3. If the key is found at middle element, the process is terminated.*
- *4. If the key is not found at middle element, choose which half will be used as the next search space.*
  - *If the key is smaller than the middle element, then the left side is used for next search.*
  - *If the key is larger than the middle element, then the right side is used for next search.*
- *5. This process is continued until the key is found or the total search space is exhausted.*

Illustration

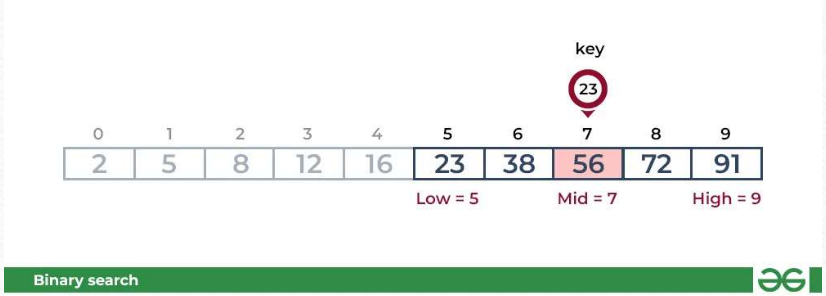


Binary search
-Analysis-

Problem

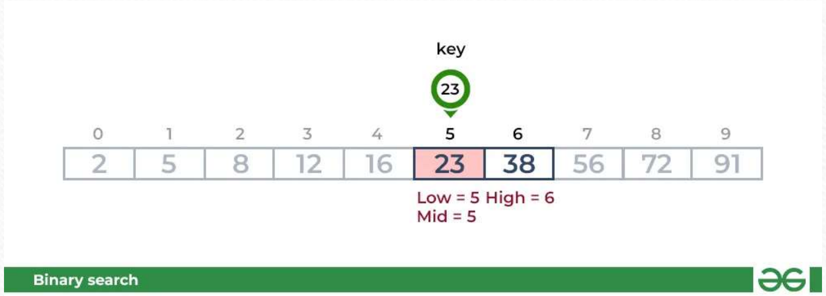


Step 1



Binary search
-Analysis-

Step 2



Analysis

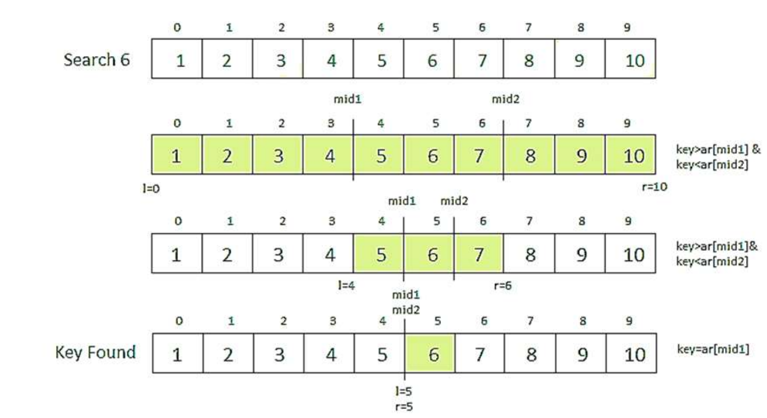
- Time complexity: O(log N)
- Auxiliary Space: O(1)

Binary search
-code-

```
arr = [2, 3, 4, 10, 40]
key = 3

index = -1
low, high = 0, len(arr)-1
while low <= high:
    mid = low + (high-low)//2

    if key == arr[mid]:
        index = mid
        break
    elif key < arr[mid]:
        high = mid - 1
    elif key > arr[mid]:
        low = mid + 1

if index >=0:
    print(f"found key: {key}, at the index: {index}")
else:
    print("Did not find the key")
```

(3) Ternary search

- similar to binary search
- we divide the given array into three parts and determine which has the key (searched element).

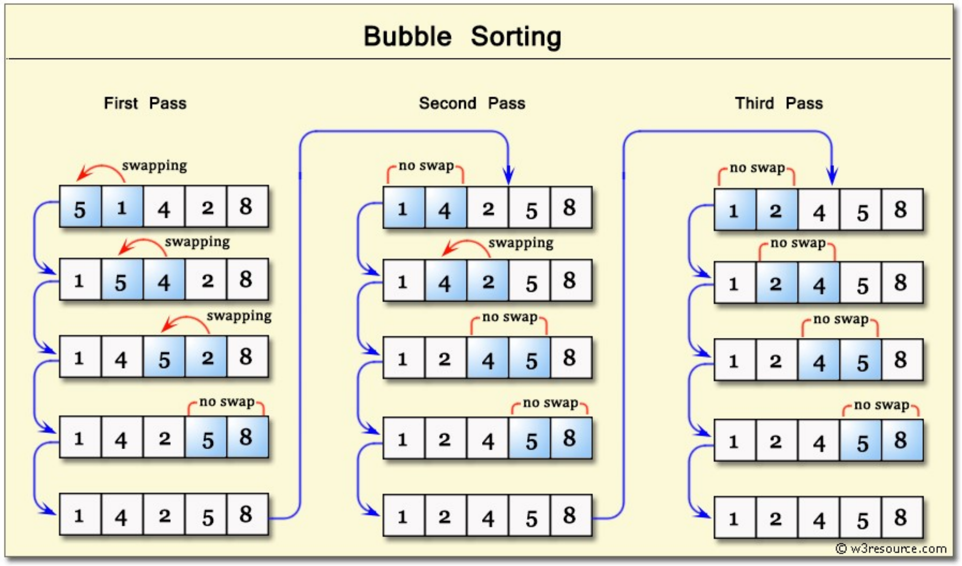


Ternary search -code-

```
arr = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
n = len(arr)
key = 10

low = 0
high = n -1
index = -1
while low <= high:
    mid1 = low + (high - low)//3
    mid2 = high - (high - low)//3

    if key == arr[mid1]:
        index = mid1
        break
    elif key == arr[mid2]:
        index = mid2
        break
    elif key < arr[mid1]:
        high = mid1 - 1
    elif key < arr[mid2]:
        low = mid1 + 1
        high = mid2 -1
    else:
        low = mid2 + 1

if index >=0 :
    print(f"the key: {key} is at element: {index}")
else:
    print(f"key: {key} not found")
```

Bubble sort
-intro-

- Bubble sort consists of multiple passes through a list
- Compares adjacent elements one by one, and swapping pairs that are out of order.
- the largest element in the list “bubbles up” toward its correct position.



Bubble sort
-code-

Code 1

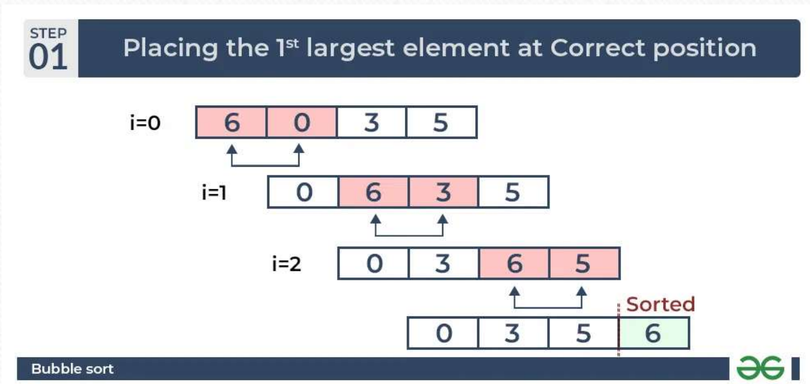
```
arr = [10, 5, 4, 20, 32, 12]
n = len(arr)

swap = True
while swap:
    swap = False
    for i in range(0, n-1):
        if arr[i] > arr[i+1]:
            swap = True
            arr[i], arr[i+1] = arr[i+1], arr[i]

print(f"sorted array is: {arr}")
```

Code 2

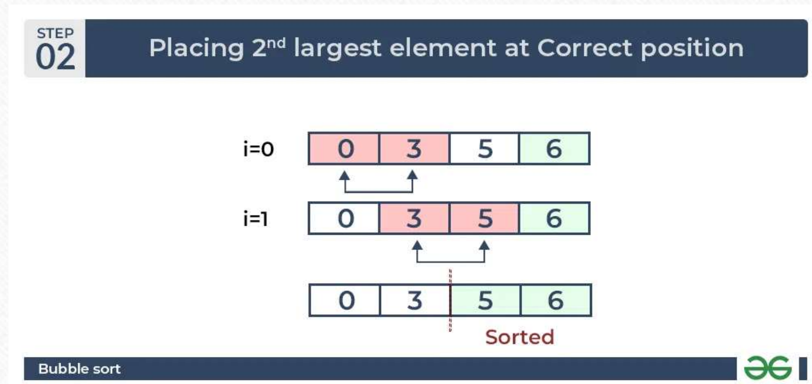
```
arr = [10, 5, 4, 20, 32, 12]
n = len(arr)

for i in range(n):

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

        if arr[j] > arr[j+1]:
            arr[j], arr[j+1] = arr[j+1], arr[j]

print(f"sorted array is: {arr}")
```

Bubble sort
-Analysis-

First Pass:

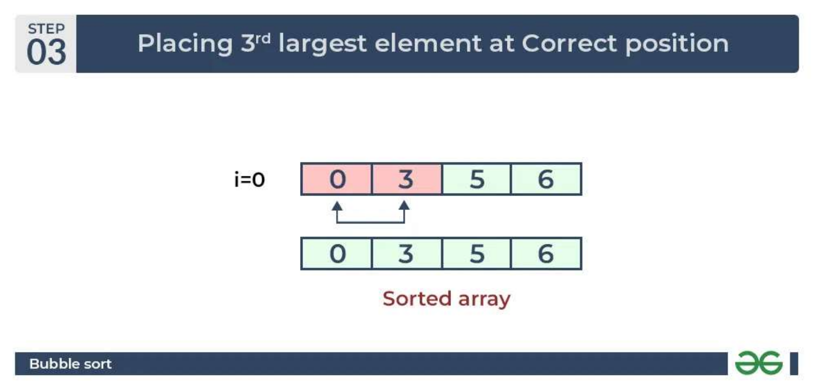


Second Pass:



Bubble sort
-Analysis-

Third pass



Analysis

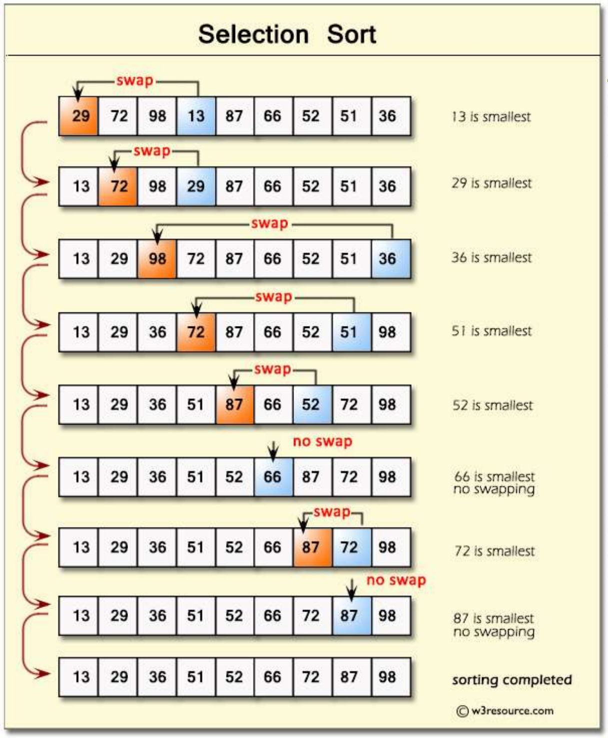
- Total no. of passes: 3
- Total no. of comparisons: 6 (4*3/2)
- Best scenario possible:
  - Input array is already sorted
- Time Complexity: O(N^2)

Detailed time complexity analysis

- Implementation of bubble sort consists of two nested for loops:
- First, the algorithm performs n - 1 comparisons, next, n - 2 comparisons, and etc until the final comparison is done.
- This comes at a total of (n - 1) + (n - 2) + (n - 3) + … + 2 + 1 = n(n-1)/2 comparisons
- Big O: focuses on how the runtime grows in comparison to the size of the input.
- To turn the above equation into the Big O complexity of the algorithm, you need to remove the constants because they don't change with the input size.
- The notation simplifies to $n^2 - n$.
- n^2 grows much faster than n, this last term can be dropped
- Bubble sort with an average- and worst-case complexity of O(n^2).

Selection sort
-intro-

- first step
  - extract the minimum element
  - Swap it with the element at index 0
- subsequent step
  - in remaining sublist, extract minimum element
  - swap it with the element at index 1
- keep the left portion of the list sorted
  - at i'th step, first i elements in list are sorted
  - all other elements are bigger than first i elements

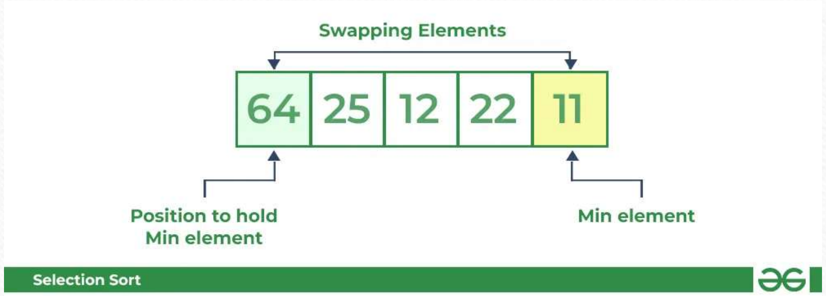


Selection sort
-code-

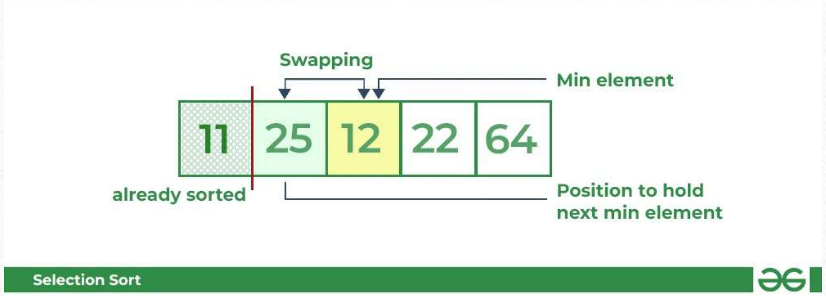
```
arr = [10, 5, 4, 20, 32, 12]
n = len(arr)

for i in range(n):
    index_min_value = i
    for j in range(i+1,n):
        if arr[index_min_value] > arr[j]:
            index_min_value = j
    arr[i], arr[index_min_value] = arr[index_min_value], arr[i]

print(f"sorted array is {arr}")
```

Selection sort
-Analysis-

First pass

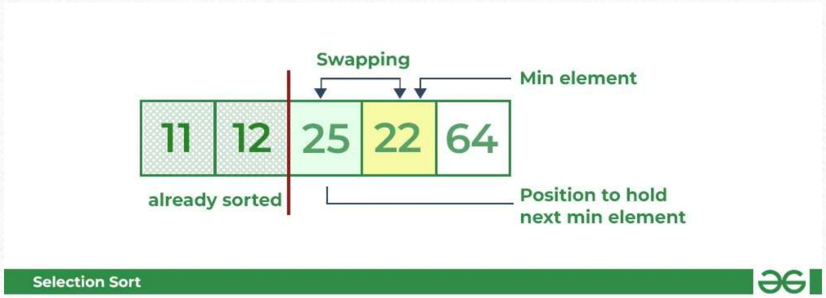


Second pass

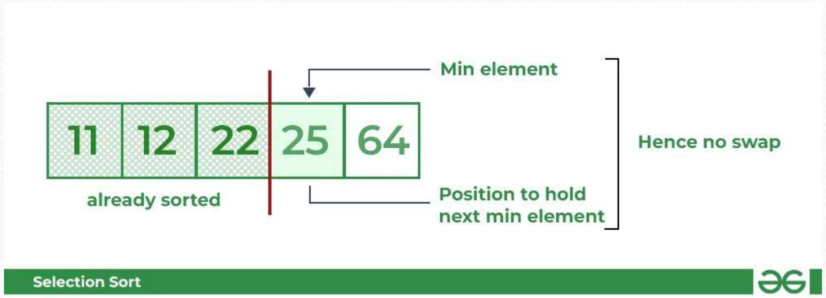


Selection sort
-Analysis-

Third pass



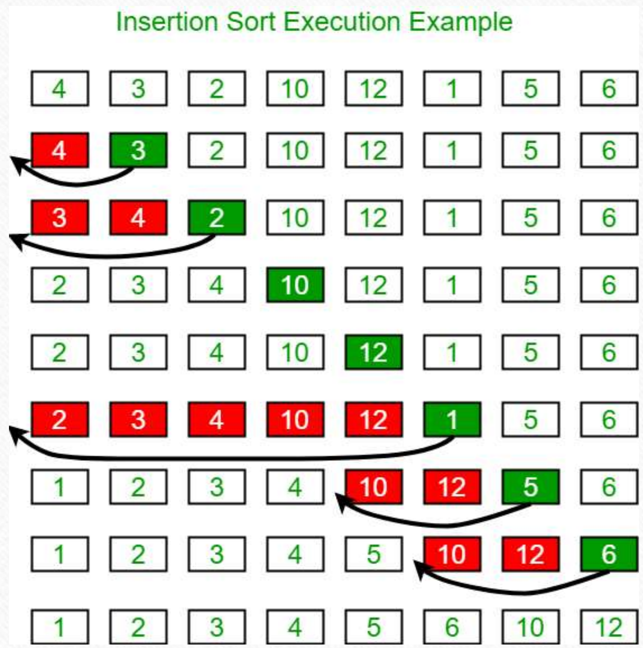
Forth pass



Time Complexity: O(N^2)

Insertion sort
-intro-

- **To sort an array of size N in ascending order:**
  - iterate over the array and compare the current element to its predecessor
    - if the element is smaller than its predecessor, compare it to the elements before.
    - Move the greater elements one position up to make space for the swapped element.



Insertion sort
-code-

Code 1

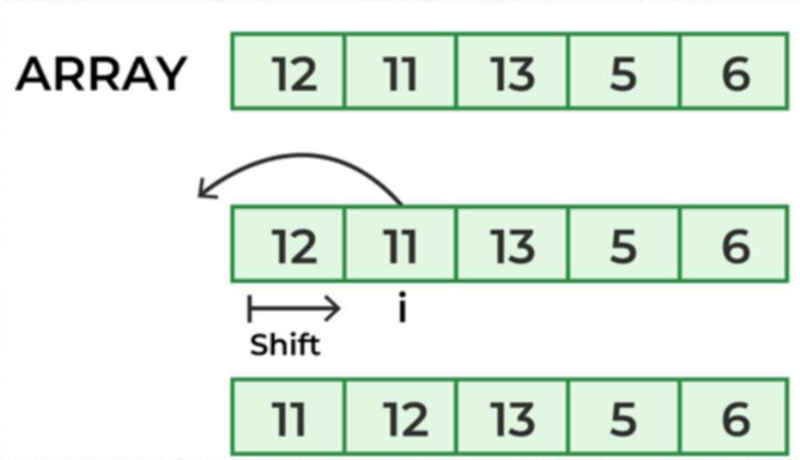
```python
arr = [10, 5, 4, 20, 32, 12]
n = len(arr)

for i in range(n):
    key = arr[i]
    for j in range(i-1,-1,-1):
        if key < arr[j]:
            arr[j+1] = arr[j]
            arr[j] = key
        else:
            break

print(f"sorted array is: {arr}")
```

Code 2

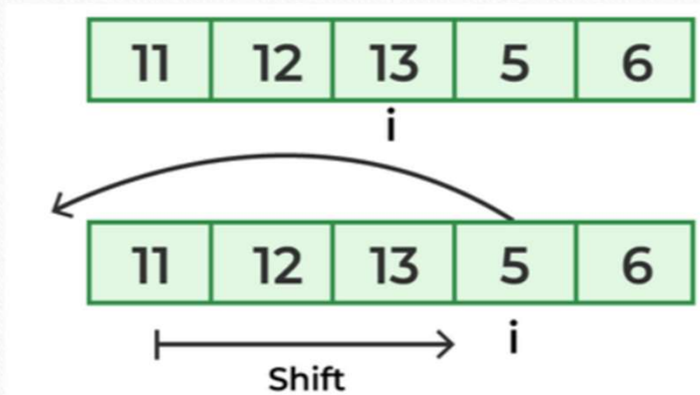
```python
arr = [10, 5, 4, 20, 32, 12]
n = len(arr)

for i in range(n):
    key = arr[i]

    j = i-1
    while j >=0 and key < arr[j] :
        arr[j+1] = arr[j]
        j -=1
    arr[j+1] = key

print(f"sorted array is: {arr}")
```

Insertion sort
-Analysis-

First pass

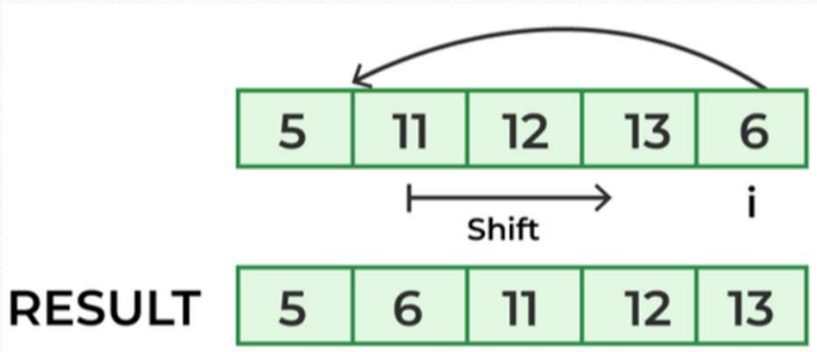


Second pass



Insertion sort
-code-

Third pass

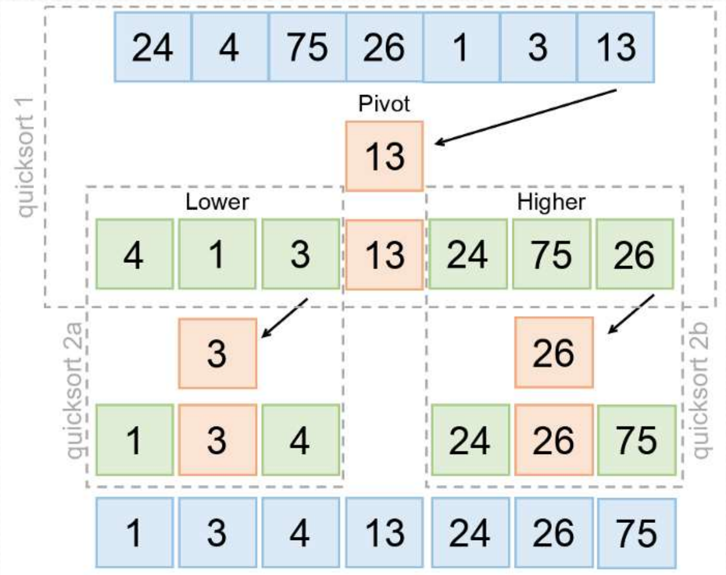


Analysis

- Time Complexity: O(N^2)

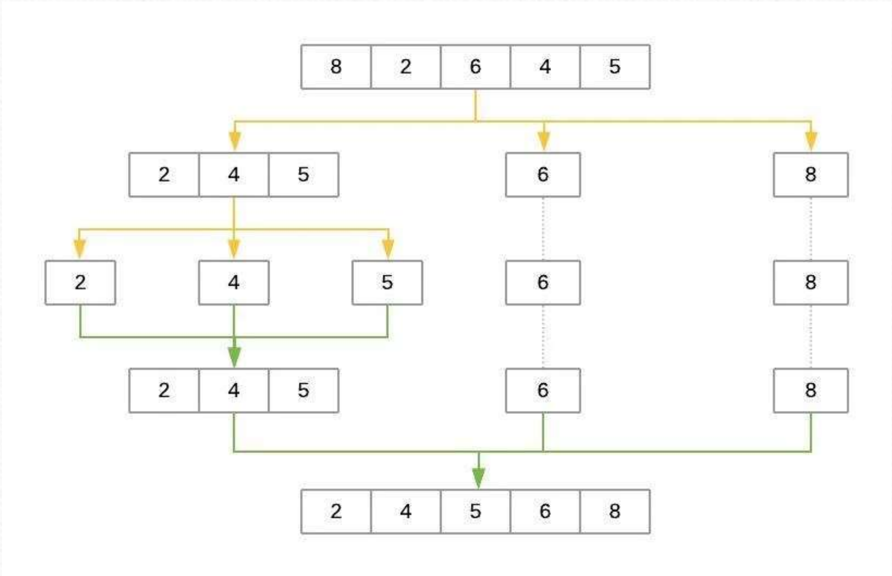
Quicksort
-intro-

- select a 'pivot' element from the array
  - First element
  - Last element
  - Randomly selected
  - Median element
- partition the other elements into two sub-arrays, according to whether they are less than or greater than the pivot.
- The sub-arrays are then sorted recursively.



https://github.com/dennisbakhuis/python10minutesaday

Quicksort
-Analysis-



https://realpython.com/sorting-algorithms-python/

Quicksort
-code-

Simple implementation (Using recursion)

```
import random

def quicksort(array):
    if len(array)<2:
        return array
    low,same,high = [], [], []
    pivot = random.choice(array)
    for item in array:
        if item < pivot:
            low.append(item)
        elif item > pivot:
            high.append(item)
        else:
            same.append(item)
    return quicksort(low) + same + quicksort(high)

arr = [10, 5, 4, 20, 32, 12]
print(quicksort(arr))
```

In place divide & conquer

```
def partition(array, low, high):
    pivot = array[high]
    i = low - 1 #pointer for the greater elements

    for j in range(low, high):
        if array[j] <= pivot:
            i = i + 1
            array[i], array[j] = array[j], array[i]
    array[high], array[i+1] = array[i+1], array[high]

    return i+1

def quicksort(array, low, high):
    if low < high:
        pivot_index = partition(array, low, high)
        quicksort(array, pivot_index+1, high)
        quicksort(array, low, pivot_index-1)

arr = [10, 5, 4, 20, 32, 12]
quicksort(arr,0,len(arr)-1)
print(f"sorted array is: {arr}")
```