

```
import time                                #5-amaliy
import matplotlib.pyplot as plt

def linear_search(arr, target):
    start_time = time.time()
    found = False
    for i in range(len(arr)):
        if arr[i] == target:
            found = True
            break
    end_time = time.time()
    return found, end_time - start_time

def binary_search(arr, target):
    start_time = time.time()
    found = False
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            found = True
            break
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    end_time = time.time()
    return found, end_time - start_time

# Sonlar ketma-ketligi
arr = [35, 69, -44, -8, 39, 4, -30, 23, -93, 9, 89, 91, -34, 83, -24]
target = 89 # Izlash uchun maqsad son

# Linear search vaqti
linear_search_times = []
for n in range(1, len(arr) + 1):
    subset_arr = arr[:n]
    found, time_taken = linear_search(subset_arr, target)
    linear_search_times.append(time_taken * 1000) # ms ga aylantiramiz

# Binary search vaqti
binary_search_times = []
for n in range(1, len(arr) + 1):
    subset_arr = sorted(arr[:n]) # Saralab olishimiz kerak
    found, time_taken = binary_search(subset_arr, target)
    binary_search_times.append(time_taken * 1000) # ms ga aylantiramiz

# Grafik chizish
plt.plot(range(1, len(arr) + 1), linear_search_times, label='Linear Search')
plt.plot(range(1, len(arr) + 1), binary_search_times, label='Binary Search')
plt.xlabel('Elementlar soni (n)')
plt.ylabel('Vaqt (ms)')
plt.title('Linear Search va Binary Search')
plt.legend()
plt.show()
```

## Linear Search va Binary Search

```
import time                                #8-amaliy Heap sort

def heapify(arr, n, i):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2

    if left < n and arr[i] < arr[left]:
        largest = left

    if right < n and arr[largest] < arr[right]:
        largest = right

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

def heap_sort(arr):
    n = len(arr)

    for i in range(n//2 - 1, -1, -1):
        heapify(arr, n, i)

    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)

    return arr

# Misol uchun berilgan sonlar
numbers = [3, 5, 34, 97, 98, 26, 21, 100, 20, 62, 55, 25, 23, 90, 2, 45, 69, 71, 83, 58, 94, 38, 72, 64, 28, 74, 57, 49, 11, 68]

start_time = time.time()
sorted_numbers = heap_sort(numbers)
end_time = time.time()

execution_time = end_time - start_time

print("Heap Sort natijasi:", sorted_numbers)
print("Dastur ishlash vaqti: %.6f s" % execution_time)

Heap Sort natijasi: [2, 3, 5, 11, 20, 21, 23, 25, 26, 28, 34, 38, 45, 49, 55, 57, 58, 62, 64, 68, 69, 71, 72, 74, 83, 90, 94, 97, 98]
Dastur ishlash vaqti: 0.000127 s
```

```
import time                                #8-amaliy

def j_sort(arr):
    n = len(arr)

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

    return arr

# Misol uchun berilgan sonlar
numbers = [3, 5, 34, 97, 98, 26, 21, 100, 20, 62, 55, 25, 23, 90, 2, 45, 69, 71, 83, 58, 94, 38, 72, 64, 28, 74, 57, 49, 11, 68]

start_time = time.time()
sorted_numbers = j_sort(numbers)
end_time = time.time()

execution_time = end_time - start_time

print("JSort natijasi:", sorted_numbers)
print("Dastur ishlash vaqti: %.6f s" % execution_time)

JSort natijasi: [2, 3, 5, 11, 20, 21, 23, 25, 26, 28, 34, 38, 45, 49, 55, 57, 58, 62, 64, 68, 69, 71, 72, 74, 83, 90, 94, 97, 98, 100]
Dastur ishlash vaqti: 0.000062 s
```

```
import matplotlib.pyplot as plt            #6-amaliy
```

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

def insertion_sort(arr):
    n = len(arr)
    for i in range(1, n):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

def calculate_complexity_score(arr):
    complexity_score = 0
    n = len(arr)
    for i in range(n):
        for j in range(i + 1, n):
            if arr[j] < arr[i]:
                complexity_score += 1
    return complexity_score

def plot_complexity_scores(scores):
    array_sizes = range(1, len(scores) + 1)

    plt.plot(array_sizes, scores, marker='o')
    plt.xlabel('Array Size')
    plt.ylabel('Complexity Score')
    plt.title('Complexity Scores for Different Array Sizes')
    plt.grid(True)
    plt.show()

arr = [35, 69, -44, -8, 39, 4, -30, 23, -93, 9, 89, 91, -34, 83, -24]

sorted_arr = arr.copy()
bubble_sort(sorted_arr)
selection_sort(sorted_arr)
insertion_sort(sorted_arr)

scores = []
scores.append(calculate_complexity_score(arr))
scores.append(calculate_complexity_score(sorted_arr))

plot_complexity_scores(scores)
```

## Complexity Scores for Different Array Sizes

```

class Node:                                #3-amaliy
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

def is_complete_or_full_binary_tree(root):
    if root is None:
        return True

    queue = []
    queue.append(root)

    is_leaf_node = False

    while len(queue) > 0:
        curr = queue.pop(0)

        if curr.left:
            if is_leaf_node:
                return False
            queue.append(curr.left)
        else:
            is_leaf_node = True

        if curr.right:
            if is_leaf_node:
                return False
            queue.append(curr.right)
        else:
            is_leaf_node = True

    return True

def construct_binary_tree(arr):
    n = len(arr)

    if n == 0:
        return None

    root = Node(arr[0])

    i = 1
    while i < n:
        if arr[i] is not None:
            root.left = Node(arr[i])
            i += 1
        if i < n and arr[i] is not None:
            root.right = Node(arr[i])
            i += 1
        root = root.left

    return root

arr = [34, 76, 78, 58, 67, 16, 49, 42, 92, 17, 1, 76, 98, 39, 29, 40, 46]

root = construct_binary_tree(arr)

print("Complete or Full Binary Tree:", is_complete_or_full_binary_tree(root))

```

Complete or Full Binary Tree: True

```

import time                                #7-amaliy

def bead_sort(arr):
    if len(arr) <= 1:
        return arr

    max_val = max(arr)
    beads = [[0] * len(arr) for _ in range(max_val)]

    for num in arr:
        for j in range(num):
            beads[j][arr.index(num)] = 1

    sorted_arr = []

    for i in range(max_val):

```

```
sorted_arr += [j + 1 for j in range(len(arr)) if beads[i][j] == 1]

return sorted_arr

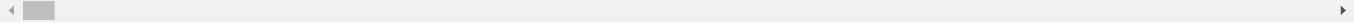
# Misol uchun berilgan sonlar
numbers = [3, 5, 3, 97, 98, 26, 21, 100, 20, 62, 55, 25, 23, 90, 2, 45, 69, 71, 83, 58, 94, 38, 72, 64, 28, 74, 57, 49, 11, 68]

start_time = time.time()
sorted_numbers = bead_sort(numbers)
end_time = time.time()

execution_time = end_time - start_time

print("Bead Sort natijasi:", sorted_numbers)
print("Dastur ishlash vaqti: %.6f s" % execution_time)

Bead Sort natijasi: [1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 1,
Dastur ishlash vaqti: 0.001453 s
```



Bead sort