১. সর্টিং (Sorting) অ্যালগরিদমসমূহ

সর্টিং মানে একটা লিস্ট বা অ্যারে-এর এলিমেন্টগুলোকে নির্দিষ্ট অর্ডারে (যেমন চেকুতি বা লেখানুসারে) সাজানো। এটি অ্যালগরিদমের ভিত্তি, কারণ অনেক ডেটা স্ট্রাকচার এতে নির্ভর করে।

১.১ Bubble Sort (বুদবুদ সর্ট)

ব্যাখ্যা: এটি সহজতম সর্টিং। লিস্টের প্রতিটি পেয়ার চেক করে সোয়াপ (swap) করে যদি ভুল অর্ডারে থাকে। বড় এলিমেন্টগুলো শেষে "বুদবুদের মতো" উঠে আসে।
সময় জটিলতা: $O(n^2)$ (খারাপ পারফরম্যান্স বড় লিস্টে)।
কখন ব্যবহার: ছোট লিস্ট বা শিক্ষার জন্য।

```python
def bubble_sort(arr):
    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]
    return arr

# উদাহরণ
my_list = [64, 34, 25, 12, 22, 11, 90]
print(bubble_sort(my_list))
```

আউটপুট:

```text
[11, 12, 22, 25, 34, 64, 90]
```

১.২ Selection Sort (সিলেকশন সর্ট)

ব্যাখ্যা: প্রতিটি পাসে লিস্টের মিনিমাম এলিমেন্ট খুঁজে প্রথম অবস্থানে রাখে, তারপর বাকিগুলোতে যায়।
সময় জটিলতা: $O(n^2)$।
কখন ব্যবহার: যখন সোয়াপ কম করতে চান।

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

# উদাহরণ
my_list = [64, 34, 25, 12, 22, 11, 90]
print(selection_sort(my_list))
```

আউটপুট:

```text
[11, 12, 22, 25, 34, 64, 90]
```

১.৩ Insertion Sort (ইনসারশন সর্ট)

ব্যাখ্যা: লিস্টকে দুই ভাগে ভাগ করে—সর্টেড অংশ এবং আনসর্টেড। প্রতিটি এলিমেন্টকে সর্টেড অংশে ইনসার্ট করে।
সময় জটিলতা: $O(n^2)$, কিন্তু আংশিক সর্টেড লিস্টে ভালো।
কখন ব্যবহার: ছোট বা প্রায় সর্টেড ডেটা।

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

# উদাহরণ
my_list = [64, 34, 25, 12, 22, 11, 90]
print(insertion_sort(my_list))
```

আউটপুট:

```text
[11, 12, 22, 25, 34, 64, 90]
```

১.৪ Merge Sort (মার্জ সর্ট)

ব্যাখ্যা: Divide and Conquer—লিস্টকে অর্ধেকে ভাগ করে সর্ট করে, তারপর মার্জ করে। স্থিতিশীল (stable) এবং দ্রুত।
সময় জটিলতা: $O(n \log n)$।
কখন ব্যবহার: বড় লিস্ট, লিঙ্কড লিস্টে ভালো।

```python
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)

def merge(left, right):
    result = []
    i = j = 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
    result.extend(left[i:])
    result.extend(right[j:])
    return result

# উদাহরণ
my_list = [64, 34, 25, 12, 22, 11, 90]
print(merge_sort(my_list))
```

আউটপুট:

```text
[11, 12, 22, 25, 34, 64, 90]
```

১.৫ Quick Sort (কুইক সর্ট)

ব্যাখ্যা: Divide and Conquer—একটা পিভট চুজ করে লিস্টকে দুই ভাগে ভাগ করে (ছোট এবং বড়), তারপর রিকার্সিভ কল।
সময় জটিলতা: গড়ে $O(n \log n)$, খারাপ ক্ষেত্রে $O(n^2)$।
কখন ব্যবহার: বড় ডেটা, ইন-প্লেস সর্টিং চাইলে।

```python
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

# উদাহরণ
my_list = [64, 34, 25, 12, 22, 11, 90]
print(quick_sort(my_list))
```

আউটপুট:

```text
[11, 12, 22, 25, 34, 64, 90]
```

১.৬ Heap Sort (হিপ সর্ট)

ব্যাখ্যা: হিপ ডেটা স্ট্রাকচার ব্যবহার করে (ম্যাক্স হিপ), রুটকে সরিয়ে হিপ রি-হিপিফাই করে।
সময় জটিলতা: $O(n \log n)$।
কখন ব্যবহার: যখন অতিরিক্ত মেমরি নেই।

```python
import heapq

def heap_sort(arr):
    heapq.heapify(arr)
    return [heapq.heappop(arr) for _ in range(len(arr))]

# উদাহরণ (পাইথনের heapq ব্যবহার করে)
my_list = [64, 34, 25, 12, 22, 11, 90]
print(heap_sort(my_list))
```

আউটপুট:

```text
[11, 12, 22, 25, 34, 64, 90]
```

১.৭ Python-এর Built-in Sorting

ব্যাখ্যা: `sorted()` ফাংশন TimSort (Merge + Insertion) ব্যবহার করে, যা দ্রুত এবং স্থিতিশীল।
সময় জটিলতা: $O(n \log n)$।
কখন ব্যবহার: সবসময়! কাস্টম কী দিয়ে সর্ট করতে `key` প্যারামিটার ব্যবহার করুন।

```python
# উদাহরণ
my_list = [64, 34, 25, 12, 22, 11, 90]
sorted_list = sorted(my_list, key=lambda x: x)  # চেকুতি
print(sorted_list)

# লিস্ট ইন-প্লেস সর্ট
my_list.sort(reverse=True)  # ডিসেন্ডিং
print(my_list)
```

আউটপুট:

```text
[11, 12, 22, 25, 34, 64, 90]
[90, 11, 22, 12, 25, 34, 64]
```

---

২. সার্চিং (Searching) অ্যালগরিদমসমূহ

সার্চিং মানে লিস্টে একটা এলিমেন্ট খুঁজে বের করা। সর্টেড লিস্টে দ্রুত হয়।

২.১ Linear Search (লিনিয়ার সার্চ)

ব্যাখ্যা: লিস্টের প্রথম থেকে শেষ পর্যন্ত চেক করে।
সময় জটিলতা: $O(n)$।
কখন ব্যবহার: আনসর্টেড লিস্ট বা ছোট ডেটা।

```python
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1

# উদাহরণ
my_list = [64, 34, 25, 12, 22, 11, 90]
print(linear_search(my_list, 25))
```

আউটপুট:

```text
2
```

২.২ Binary Search (বাইনারি সার্চ)

ব্যাখ্যা: লিস্ট সর্টেড হতে হবে। মিডল চেক করে, লেফট বা রাইটে যায়।
সময় জটিলতা: $O(\log n)$।
কখন ব্যবহার: সর্টেড লিস্টে দ্রুত সার্চ।

```python
def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1

# উদাহরণ (প্রথমে সর্ট করুন)
my_list = sorted([64, 34, 25, 12, 22, 11, 90])
print(binary_search(my_list, 25))
```

আউটপুট:

```text
3
```

২.৩ Hashing (হ্যাশিং) বা Dictionary Search

ব্যাখ্যা: পাইথনের dict ব্যবহার করে $O(1)$ গড় সময়ে সার্চ। কী-ভ্যালু পেয়ারে কাজ করে।
সময় জটিলতা: গড়ে $O(1)$, খারাপ ক্ষেত্রে $O(n)$।
কখন ব্যবহার: ফ্রিকোয়েন্ট সার্চ, যেমন ডাটাবেস।

```python
# উদাহরণ
my_dict = {'apple': 1, 'banana': 2, 'cherry': 3}
target = 'banana'
if target in my_dict:
    print(my_dict[target])
else:
    print("Not found")
```

আউটপুট:

```text
2
```

২.৪ Python-এর Built-in Search

ব্যাখ্যা: `in` অপারেটর লিনিয়ার সার্চ করে। সর্টেড লিস্টে `bisect` মডিউল ব্যবহার করুন Binary-এর জন্য।

```python
import bisect

# উদাহরণ
my_list = sorted([64, 34, 25, 12, 22, 11, 90])
index = bisect.bisect_left(my_list, 25)
print(index if index < len(my_list) and my_list[index] == 25 else "Not found")
```

আউটপুট:

```text
3
```

শেষ কথা এবং টিপস

- **তুলনা**: সর্টিং-এ $O(n \log n)$ (Merge/Quick) বড় ডেটার জন্য ভালো; $O(n^2)$ শুধু ছোটের জন্য। সার্চিং-এ Binary সবচেয়ে দ্রুত যদি সর্টেড হয়।
- **প্র্যাকটিস**: এই কোডগুলো রান করে নিজে পরীক্ষা করুন। পাইথনে `timeit` মডিউল দিয়ে পারফরম্যান্স মাপুন।
- **পরবর্তী ধাপ**: রিকার্সন বা জেনারেটর নিয়ে শিখুন। কোনো প্রশ্ন থাকলে জিজ্ঞাসা করুন!