DSC 40B

Lecture 15 : Hashing Good and Bad

Fast Algorithms with Hash Tables

Faster Algorithms

- Hashing is a **super** common trick.
- The "best" solution to interview problems ***often*** involves hashing.

Example 1: The Movie Problem

- You're on a flight that will last D minutes.
- You want to pick two movies to watch.
- Find two whose durations sum to **exactly** D.

Recall: Previous Solutions

- Brute force: $\Theta(n^2)$.
- Sort, use sorted structure: $\Theta(n \log n) + \Theta(n)$.
- Theoretical lower bound: $\Omega(n)$?
- Can we speed this up with **hash tables**?

Idea

- To use hash tables, we want to frame problem as a **membership** query.

Example

- Suppose flight is 360 (*D*) minutes long.
- Suppose first movie is fixed: 120 minutes.
- Is there a movie lasting (360 - 120) = 240 minutes?

```python
def optimize_entertainment_hash(times, D):
    hash_table = dict()
    for i, time in enumerate(times):
        hash_table[time] = i

    for i, time in enumerate(times):
        target = D - time
        if target in hash_table:
            return i, hash_table[target]
```

Example 2: Anagrams

- Two strings w_1 and w_2 are **anagrams** if the letters of w_1 one can be *permuted* to make w_2.

Examples of anagrams

- abcd / dbca
- listen / silent
- sandiego / doginsea

Problem

- Given a collection of n strings, determine if any two of them are anagrams.
- Design an efficient algorithm for solving this problem. What is its time complexity?

Solution

- Let's turn this into a *membership* query.
- **Trick**: two strings are anagrams if

```
sorted(w₁) == sorted(w₂)
```

```python
def any_anagrams(words):
    seen = set()
    for word in words:
        w = sorted(word)
        if w in seen
            return True
        else:
            seen.add(w)
```

What is the worst time complexity?

A: Constant

B: n

C: $n \log n$

D: n^2

Hashing Downsides

- Problem must involve ***membership*** query.

Example: The Movie Problem

- You're on a flight that will last D minutes.
- You want to pick two movies to watch.
- Find two whose added durations is **closest** to D.

Hashing Downsides

- **No locality**: similar items map to different bins.
- There is no way to quickly query entry closest to given input.

Locality

Memory: To access a value, we must know its **address** .

Sequences

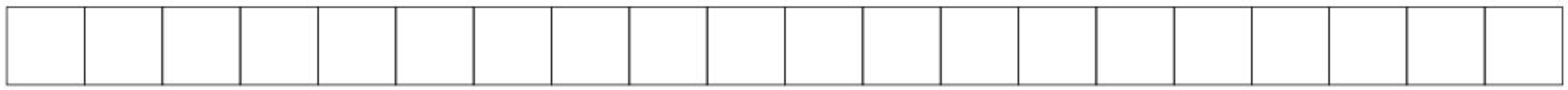
- How do we store an **ordered** (sorted) sequence?
    - E.g. 10, 20, 30, 80, 95, 96, 190

Arrays

Good and Bad

Arrays

- Store elements **contiguously (next to each other)**:
  - E.g. 10, 20, 30, 80, 95, 96, 190



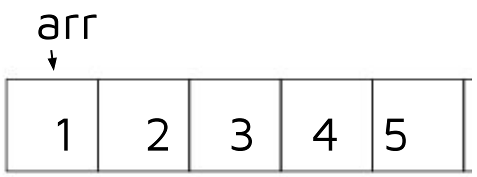
- NumPy arrays are…arrays

Allocation

- Memory is *shared* resource.
- A chunk of memory of fixed size has to be reserved (**allocated**) for the array.
- The size has to be known **beforehand**.

Arrays

- To access an element, we need its **address**.
- **Key Idea**: Address are easily calculated.
  - int arr [] = {1,2,3,4,5}
  - For kth element:
    - Address(arr[k]) = Address of the first + k x Size of each element
- Therefore, arrays support $\Theta(1)$ -time access



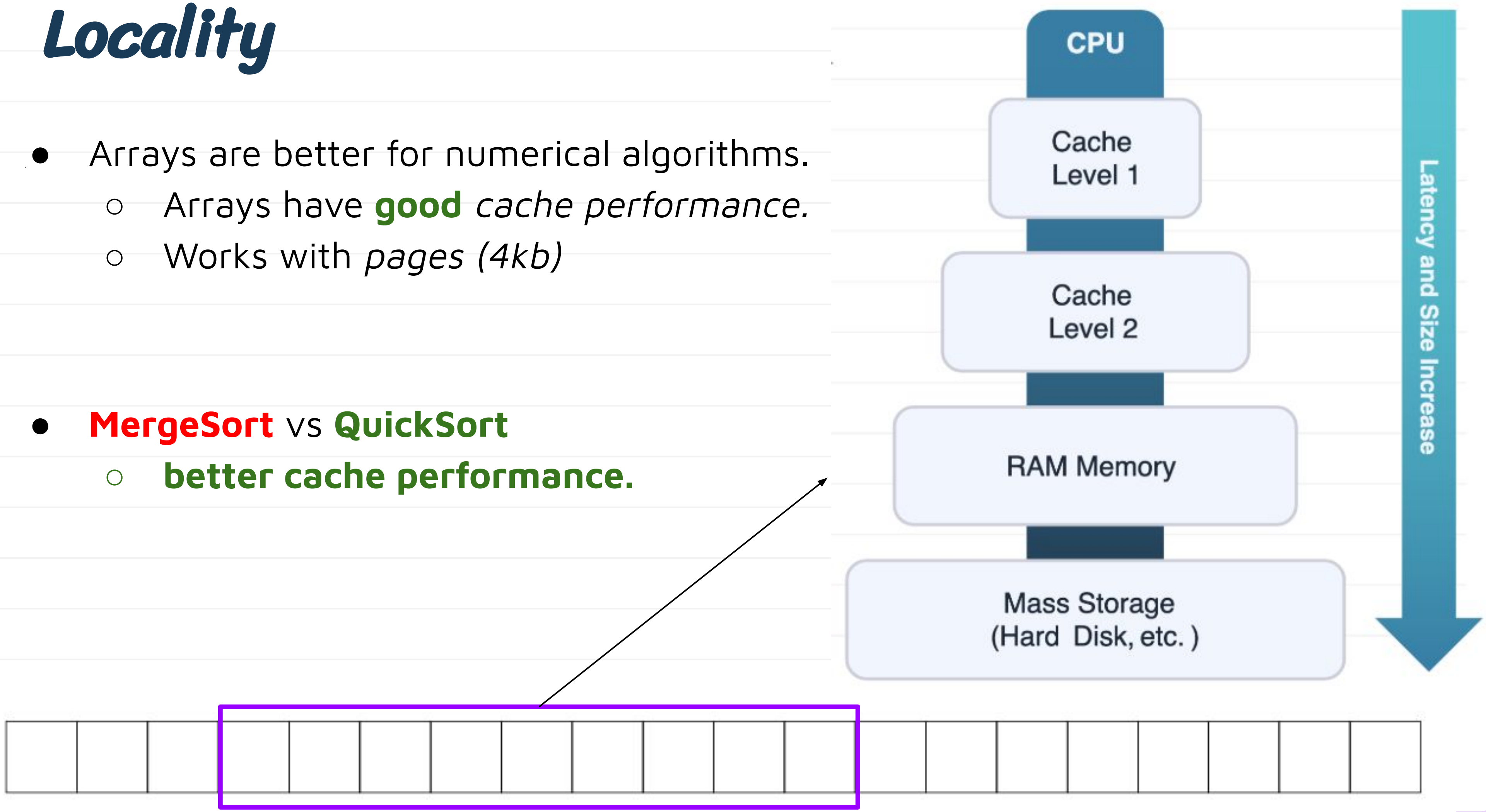
Downsides of Arrays

- **Homogeneous**: every element must be same size.
- To **resize** the array, a totally **new chunk of memory** has to be found; old values copied over.

Locality

- Arrays are better for numerical algorithms.
  - Arrays have **good** *cache performance.*
  - Works with *pages (4kb)*

- **MergeSort** vs **QuickSort**
  - **better cache performance.**



Hashing Downsides

- **No locality**: similar items map to different bins.
- There is no way to quickly query entry closest to given input.

Example: Number of Elements

- Given a collection of n numbers and two endpoints, a and b, determine how many of the numbers are contained in $[a, b]$.
- Not a membership query.
- **Idea**: **sort** and use *modified* binary search.

Hash Table Drawbacks

Hashing ***Downsides***

- No locality: similar items map to different bins.
- But we often want similar items at the same time.
- Results in many **cache misses**, **slow**.
- Memory overhead.

Hash Tables vs. BSTs

- **Hash Table**: $\Theta(1)$ insertion, query (expected time).
- **BST**: $\Theta(\log n)$ insertion, query (if *balanced*).
- Why ever use a BST?

Hash Tables vs. BSTs

- Hash tables keep items in arbitrary order.
- **Example**: how many elements are in the interval [3, 23]?
- **Example**: what is the min/max/median?
- BSTs win when order is important.

Thank you!

Do you have any questions?

CampusWire!