FS12

Week 06
Sorting Algorithms I

Mikhail Masyagin
Daniil Devyatkin

Best Sorting Asymptotic Complexity I

- We can parallelize sort, use hardware features such as vectorized calculations & instructions parallelism to optimize our sorting algorithm. We can use special ASICs, designed for sorting only!!!
- However, slowest hardware with better asymptotic complexity sorting algorithm will beat any super-computer with worse asymptotic complexity sorting algorithm. It may take millions or billions of numbers to achieve this outperformance, it can be only in theory sometimes, yet possible.

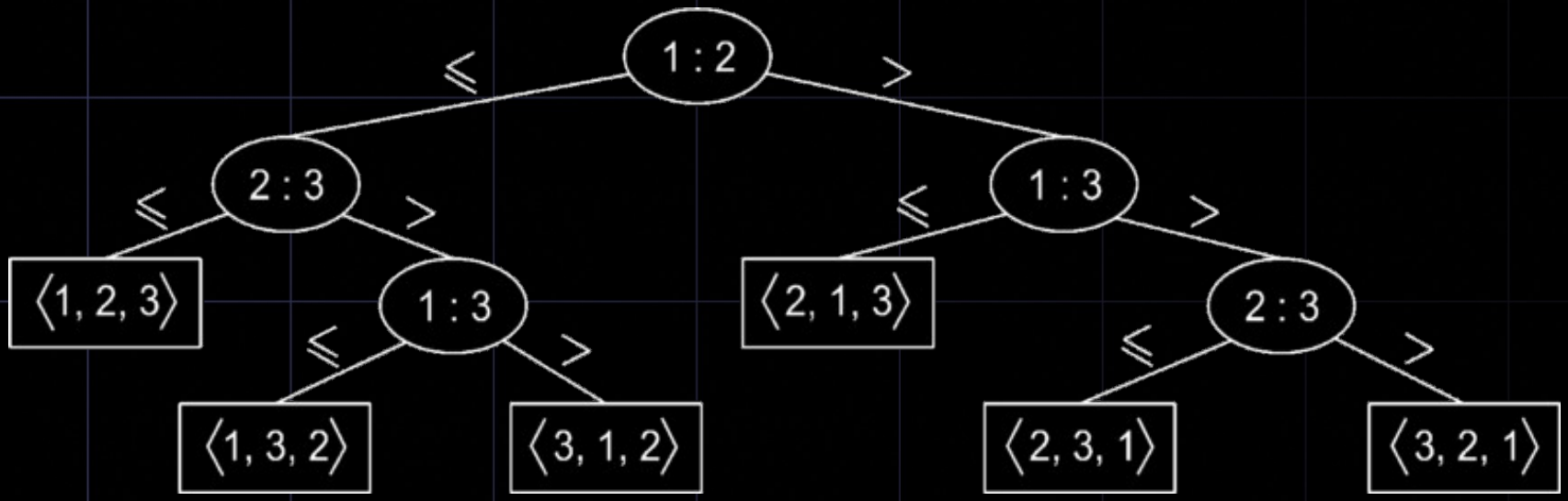
Best Sorting Asymptotic Complexity II

- Let’s try to find best Asymptotic Complexity for sorting algorithm with comparator. “With comparator” means that sorting algorithm can only compare values in array using < and > (<= or >= operators) without diving deep into their structure.
- We can do it by interpreting sorting algorithm as walkthrough the binary tree.

Best Sorting Asymptotic Complexity III

- Theorem: in the worst case every sorting algorithm with comparator uses **Ω(Nlog(N))** comparisons, where **N** is the number of sorted elements.
- Proof:
  - Lets assume that we have array of length **N**, so we have **N!** placements of elements in it.
  - Each placement is the leaf of binary tree of “sorts”.
  - It’s easy to show, that height of tree with height **H** is less than **2^H**.

Best Sorting Asymptotic Complexity IV



Best Sorting Asymptotic Complexity V

- So N! <= L <= 2^H, where L is the number of leaves.
- Lets apply log to this equation:
- log2(2^H)>=log2(L)>=log2(N!)=log2(1)+log2(2)+...+log2(N)=(N/2)log2(N/2)=(N/2)(log2(N)-1)=
  =Ω(Nlog(N))

Quick Sort I

- History: **qsort** or Hoare's sort is one of the most known sorting algorithms in the world. It was invented by Charles Hoare in 1960 during his work at… MSU!
- It is one of the fastest (by asymptotic complexity **Ω(Nlog(N))**) and simplest sorting algorithms.
- For today it is the most common sorting algorithm in STLs of many programming languages. For example, C.

Quick Sort II

- Basic algorithm idea:
  - Choose **“support element“** from array.
  - Partition: reorder elements in the array, so elements, less than **“support element“**, will be before it, and other elements (bigger than **“support element“**) after it.
  - Recursively apply procedure from two previous steps to subarrays, splitted by **“support element“**.

Quick Sort III

- The most complicated task in **qsort** is choose of **“support element“**.
- In early implementations **“support element“** always was the first element of array, but it led to worse asymptotic complexity on sorted arrays.
- In current implementations **“support element“** is chosen randomly or by special heuristics. For example, in Java is **“support element“** chosen from 5 array elements: first, middle, last and middle elements between first & middle, last & middle.

Quick Sort IV

- Partition procedure for middle “support element:

```
def part(a: List[int],l: int,r: int):
    v = a[(l + r) // 2]
    i, j = l, r
    while i <= j:
        while a[i] < v: i += 1
        while a[j] > v: j -= 1
        if i >= j: break
        swap(a[i], a[j])
        i, j = i + 1, j - 1
    return j
```

Quick Sort V

```
def qsort(a: List[int], l: int, r: int):
    if l <= r:
        q = part(a, l, r)
        qsort(a, l, q)
        qsort(a, q + 1, r)
```

Quick Sort VI

qsort nuances:

- Ω(N) = Nlog(N), O(N) = N^2;
- unstable;
- highly utilizes stack, because of recursion;
- possible optimizations:
  - use heap-based programmers stack to avoid recursion;
  - use insertion sort for small sub-arrays;
  - can be parallelized easily;
- can be written without any additional memory;
- divide-and-conquer principle.

Heap Sort I

- History: **hsort** or Williams' sort was invented by John Williams in 1964.
- Algorithm: on previous lecture we learned about heap data structure and that we can build heap from randomized array elements order via **O(n)** comparisons. Let's build Max heap from our array. Then we can remove elements from heap's root one by one and reorder heap each time, when new root element is removed. Removed elements is placed in the end of the array, so heap size is decreased.

Heap Sort II

```
from heapq import heappop, heappush

def hsort(a: List[int]):
    # TODO: rewrite to O(1) extra memory
    heap, ordered = [], []
    for e in a:
        heappush(heap, e)
    while heap:
        ordered.append(heappop(heap))
    return ordered
```

Heap Sort III

hsort nuances:

- **Ω(N) = Nlog(N)**, **O(N) = Nlog(N)**;
- unstable;
- can’t be parallelized easily;
- can be written without any additional memory.

Merge Sort I

- History: **msort** was invented by John von Neumann in 1945.
- Algorithm:
  - Split array into two parts with equal sizes.
  - Reorder these subarrays using same algorithm.
  - Merge two subarrays into new sorted array.

Merge Sort II

```
def msort(a: List[int]):
    if len(a) in [0, 1]:
        return a
    a = msort(a[:len(a) // 2])
    b = msort(a[len(a) // 2:])
    return merge(a, b)
```

Merge Sort III

```
def merge(a: List[int], b: List[int]):
    c = []
    while (len(a) != 0) and (len(b) != 0):
        if a[0] < b[0]:
            c.append(a[0])
            a.remove(a[0])
        else:
            c.append(b[0])
            b.remove(b[0])
    return c + (b if not a else a)
```

Merge Sort IV

msort nuances:

- **Ω(N) = Nlog(N)**, **O(N) = Nlog(N)**;
- stable;
- highly utilizes stack, because of recursion;
- possible optimizations:
  - use heap-based programmers stack to avoid recursion;
  - use insertion sort for small sub-arrays;
  - can be parallelized easily;
- can't be written without additional memory;
- divide-and-conquer principle.