A Computational Comparison of Sorting Algorithms

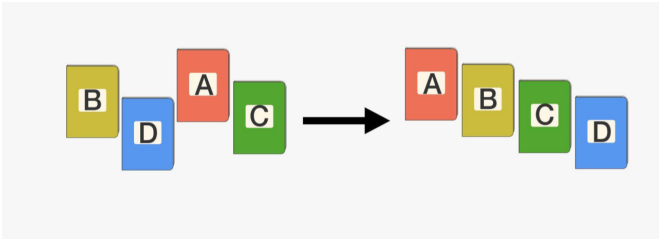
Group 13
GUPTA, Ayush
BSc in MATH-CS track at HKUST

Brief outline

- Introduction
- Methods used
- Findings
- Conclusion
- Takeaway
- Q & A

Introduction

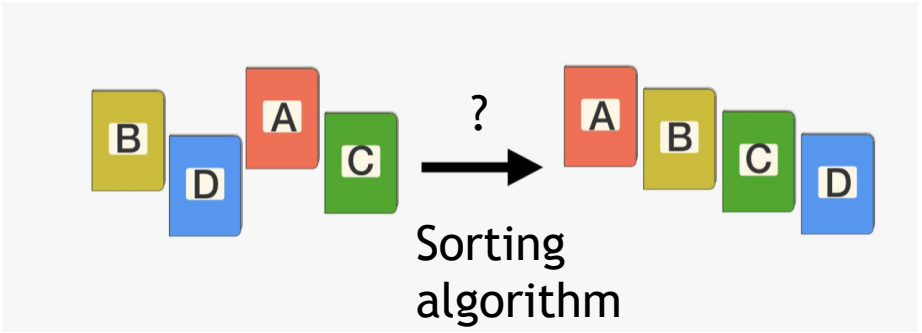
- What is sorting?
  - [3,1,2] -> [1,2,3] (ascending order)
  - BDAC -> ABCD



- Why is sorting important?
  - Easier to
    - Understand the data
    - Extract useful information

Introduction

- Sorting algorithms



- Most popular:

1. Bubble sort
2. Insertion sort
3. Merge sort
4. Quick sort

Introduction

- Basic idea

1. Bubble sort – repeatedly swap adjacent if in wrong order. Ex: [4,2] -> [2,4]
2. Insertion sort – selectively sorts first i and inserts the (i+1)th in correct place
3. Merge sort – divides array at middle, then sorts the 2 halves and merges
4. Quick sort – Breaks the array at a random place, then sorts the 2 halves and merges

Introduction

- Bubble sort python code

```
def bubbleSort(alist):
    for passnum in range(0, len(alist)-1,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:          # Example: In [4,2], 4>2. thus swap 4 and 2 -> [2,4]
                temp = alist[i]              # Compares each pair of adjacent elements
                alist[i] = alist[i+1]
                alist[i+1] = temp

alist = [50,20,90,40]
bubbleSort(alist)
print(alist)
```

Introduction

- Bubble sort no of operations (compare with other algorithms)
- Input alist = [50,20,90,40]
- Set 1:
  1. Compare 50 with 20: swap, alist = [20,50,90,40]
  2. Compare 50 with 90: alist = [20,50,90,40]
  3. Compare 90 with 40: swap -> [40,90], alist = [20,50,40,90]

     Largest element in their correct place
- Set 2:
  4. Compare 20 with 50, alist = [20,50,40,90]
  5. Compare 50 with 40, swap, alist = [20,40,50,90]

     Two largest elements in their correct places
- Set 3:

Introduction

- Bubble sort no of operations
- Input alist = [50,20,90,40]
- Set 3: (Current alist = [20,40,50,90])

  6. Compare 20 with 40, alist = [20,40,50,90]

     Three largest elements in their correct places, thus all elements sorted

- Bubble sort took 6 operations
- Actually for n elements, it takes n choose 2 operations
  - (n-1)+(n-2)+(n-3)+…+3+2+1 = (n-1)(n)/2 = n choose 2

Introduction

- Number of operations: N(operations)
- Number of elements to be sorted: n

- Previous researchers:

1. Consider n tends to infinity
2. Consider worst running time

- Real life:

1. n is never infinity
2. Consider average running time

This creates a research gap

Introduction

- Previous researches:

1. Bubble sort, Insertion sort: O(n^2)
2. Merge sort, Quick sort: O(nlog(n))

3. For n tends to infinity
4. Worst running time

Introduction

- My research: (to minimize current research gap)

1. Practically compute no of operations for Bubble sort, Insertion sort, Merge sort, Quick sort

2. For n not infinity, instead n = 100 to 1000000

3. Average running time

Methods Used

- My research:

1. Python programming language

2. Generate m arrays of n elements, from 1 to n (m is generally 1000, n is 100 to 1 million)
    1. Simple example: 2 arrays of 3 elements from 1 to 3: [1,2,3] & [2,1,3]

3. Run the algorithms one by one, note the average no of operations

Methods Used

- My research:

1. Try this for different values of n = 100,1000,10000,100000,1000000

2. Plot and interpret the results

Methods Used

- My research:

1. Compare the results with previous researchers results

2. Compare the sorting algorithms among themselves

Findings

- My research:

1. Bubble sort takes n(n-1)/2 as N(operations).
2. Insertion sort takes approximately n(n-1)/4 as N(operations).
3. Whereas previous researchers used n^2 by assuming n tends to infinity.

N(operations) = Average no. of operations

n = Number of elements to be sorted

Findings

- My research:
- Number of operations:

n (No of elements to sort)	Merge sort (MS)	Quick sort (QS)	nlog(n) (Previous researches)	QS/MS
100	14.8%	97.4%	664	6.6
1000	10%	110.6%	9966	11
10000	7.5%	117.2%	132877	15.6
100000	6%	121.8%	1660964	20.2
1000000	5%	123.2%	19931568	24.6

- Rounded off to one decimal

Conclusion

- Insertion sort (IS) is twice better than Bubble sort (BS)

- Merge sort (MS) is better than Quick sort (QS)
- MS and QS are better than IS and BS

- Very useful method for practical comparison
  - Offers valuable insights
  - More applicable to real world

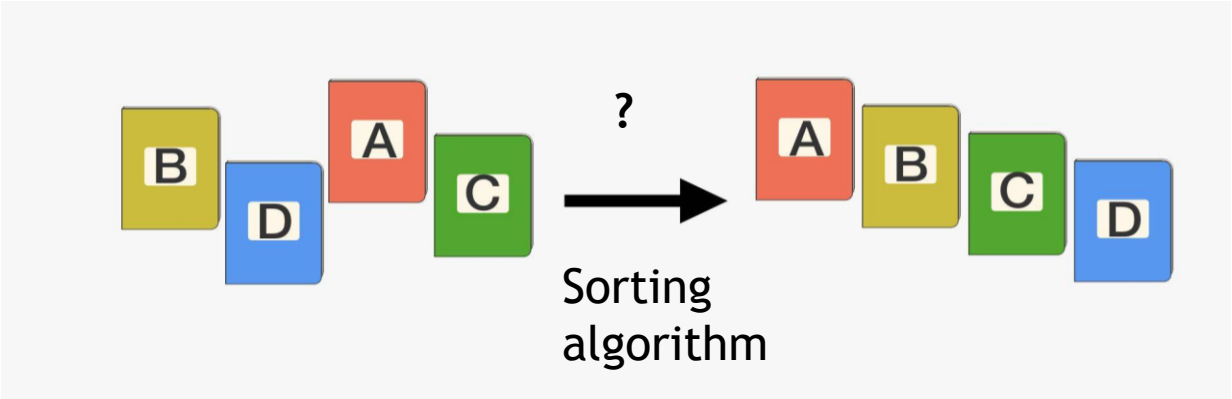
Conclusion

- Limitations:
  - Only compared up to n = 1 million. Memory error in Python after that.
  - Other case-specific ways to implement algorithms
  - Sometimes, other parameters than N(operations) can also matter

- Further research:
  - Compare other sorting algorithms using methods from this research
  - Try other ways to implement sorting algorithms
  - Try cases where n > 1 million

Takeaway

Usefulness of method of comparison:
Computational>Theoretical



References

1. K. Ali, A Comparative Study of Well Known Sorting Algorithms, Retrieved from http: //www.ijarcs.info/index.php/Ijarcs/article/view/2903/2886

2. https://www8.cs.umu.se/kurser/TDBA77/VT06/algorithms/BOOK/BOOK/NODE30.HTM

3. COMP 3711 Lecture notes, Fall'18, Retrieved from https://course.cse.ust.hk/comp3711/schedule.html

Thank You!

Q & A