Algorithms and Complexity

Sorting Algorithms

Lecturer:Asst.Prof.Dr. Alaa Ahmed Abbood
Lecture 6.
Class 2nd .
Time: 8:30-10:30
Department: Businesses Information Technology (BIT)

Outline

Sorting Algorithms

1- Quick Sort

2-Shell Sort

3- Homework

DIVIDE AND CONQUER ALGORITHM

The divide and conquer strategy solves a problem by :

1. Breaking into sub problems that are themselves smaller instances of the same type of problem.

2. Recursively solving these sub problems.

3. Appropriately combining their answers.

Quick Sort

- **Quicksort** is an efficient sorting algorithm, serving as a systematic method for placing the elements of an array in order. Developed by Tony Hoare in 1959 and published in 1961, it is still a commonly used algorithm for sorting. Quicksort is based on the divide-and conquer approach. Quicksort divides the array for two sub arrays according to their values.

Example: Quick Sort

7	2	1	6	8	5	3	4

Choose a random number call it pivot Let's say the pivot=4

Then, All elements less than pivot on Left and all elements more than pivot on right.

2	1	3	4	8	5	7	6

Cont..

We divided the array for two sub arrays depending on pivot

A 

A 

Then we need to sort the two subarrays

Let's start with A 

- Choose a Pivot=3
- No change because all elements less than pivot on left

Cont..

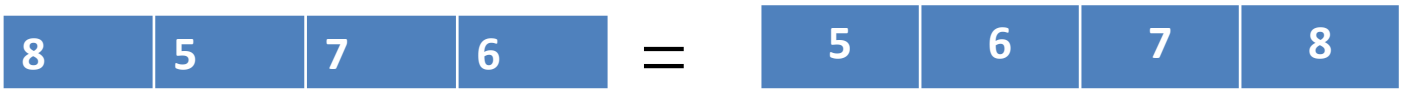
After that we work on array index from zero to one 

The pivot is 1, so the resulted segment of array is 

- When we have one element after partition the array, the recursion is stopped and the array is sorted.

Cont..

We sort the second sub array from index 4 to index 7 by the same way



Quick Sort Algorithm

```
QuickSort( A, Start, End)
{
If(Start<End)
{
Pindex←Partition (A, Start, End)
QuickSort( A, Start, Pindex-1)
QuickSort( A, Pindex+1, end)
}
}
```

Cont..

Notes:

- We need Start and End because we are working on the same array

- Partition function divide the array for two sub arrays and return the index of Pivot

Partition Function

```
Partition( A, Start, End)
{
        Pivot←A[end]
        Pindex←Start
        For(i=Start to end-1)
{
        If (a[i]<=Pivot )
{
Swap(A[i], A[Pindex])
Pindex=Pindex+1
}
}
```

Note: The idea is to push all elements less than the Pivot to the left of Pindex

Analysis of QuickSort

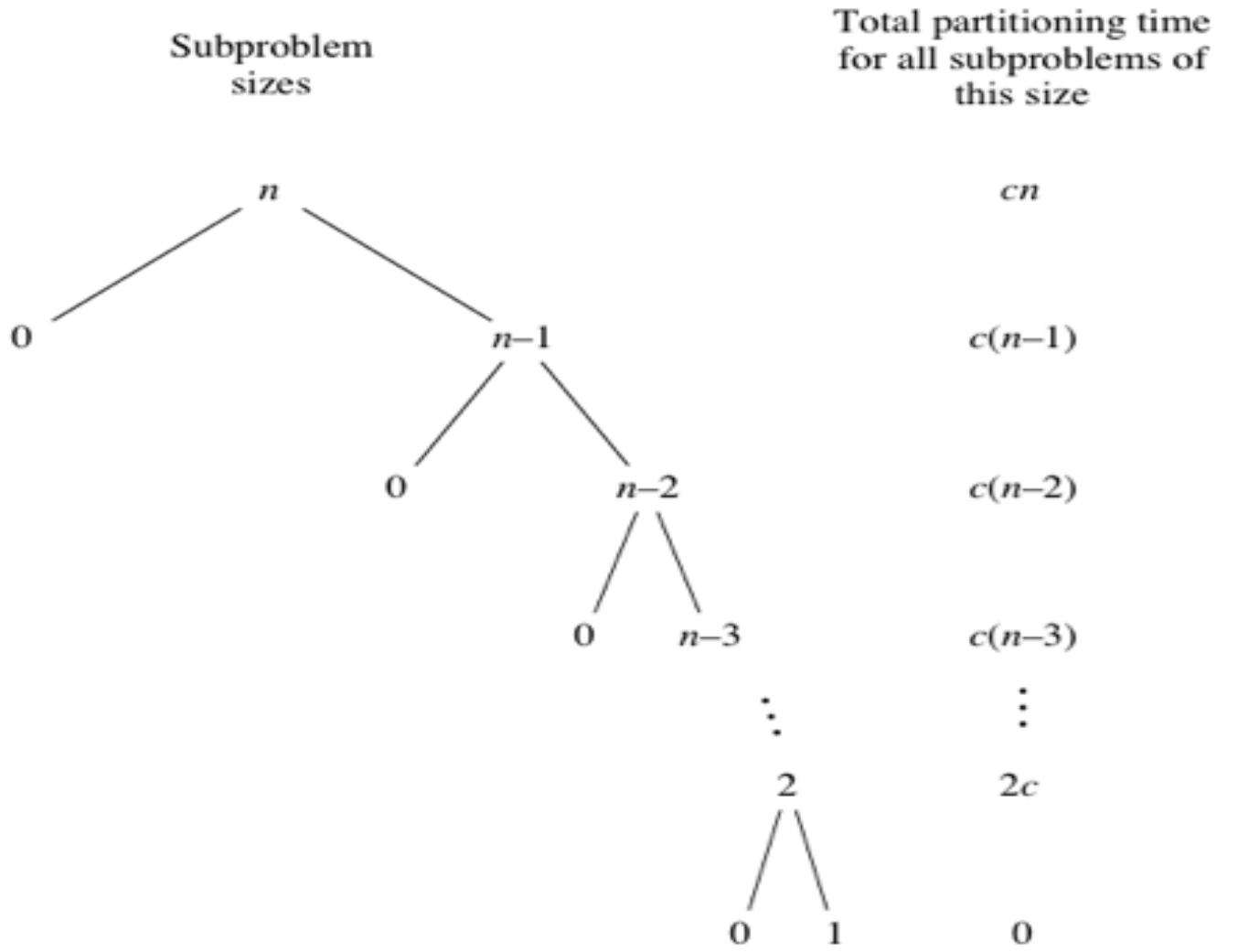
- How is it that quicksort's worst-case and average-case running times differ?
- Let's start by looking at the worst-case running time. Suppose that we're really unlucky and the partition sizes are really unbalanced. In particular, suppose that the pivot chosen by the partition function is always either the smallest or the largest element in the n nn-element subarray.

- Then one of the partitions will contain no elements and the other partition will contain n- 1elements—all but the pivot. So the recursive calls will be on subarrays of sizes 0 and n-1.

Worst-case running time

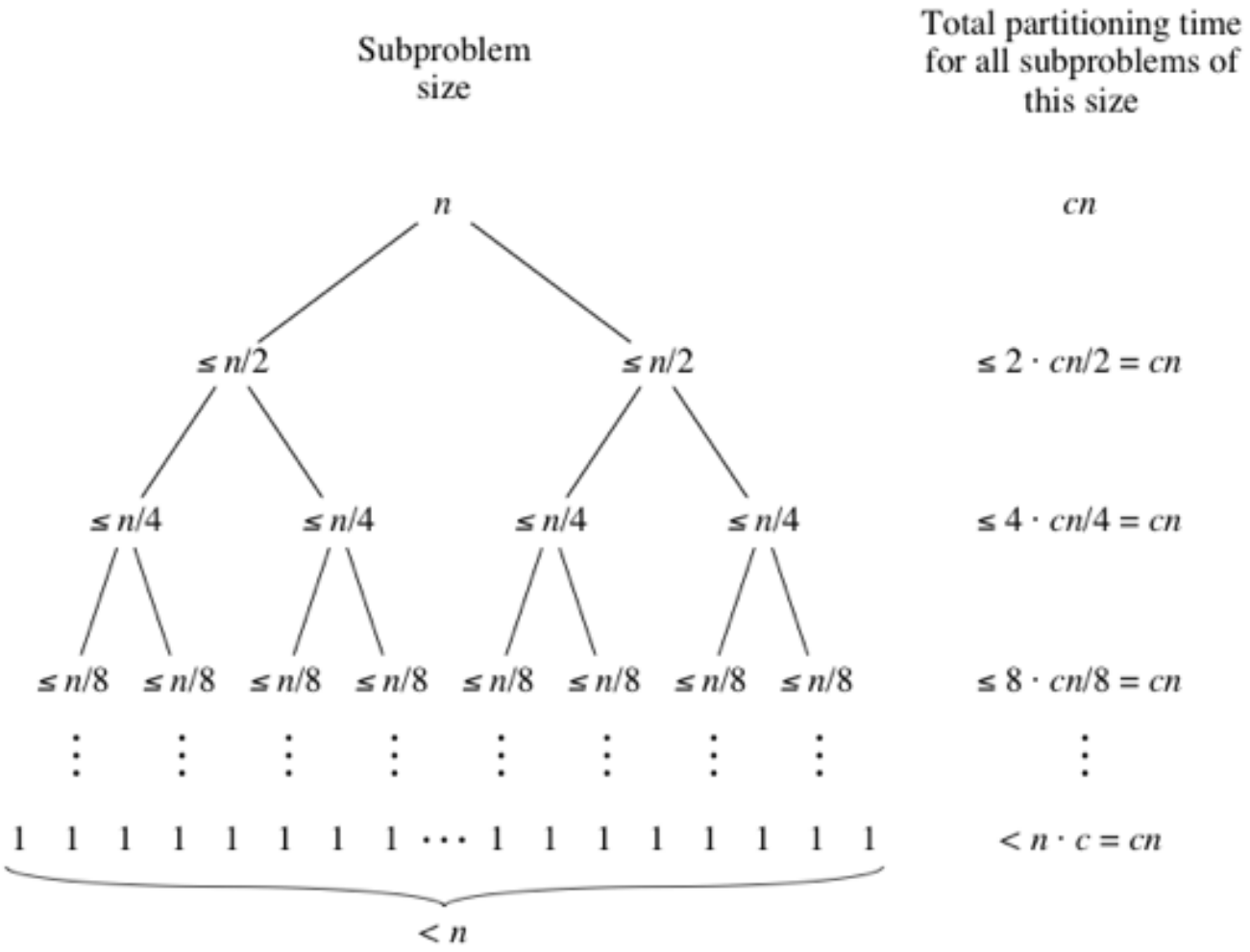
- When quicksort always has the most unbalanced partitions possible, then the original call takes cn, n time for some constant c, the recursive call on n-1 elements takes c(n-1) time. The recursive call on n-2 elements takes $c(n-2)$,and so on. Here's a tree of the subproblem sizes with their partitioning times:

Cont..


Subproblem sizes
Total partitioning time for all subproblems of this size
n
cn
0
n–1
c(n–1)
0
n–2
c(n–2)
0
n–3
c(n–3)
2
2c
0
1
0

Best-case running time

Quicksort's best case occurs when the partitions are as evenly balanced as possible: their sizes either are equal or are within 1 of each other.



Cont..

Time Complexity:

Worst case: $O(N^2)$

Best Case: O(N(logN)

Average case: O(N(logN)

Outline

Sorting Algorithms

1- Quick Sort

2-Shell Sort

3- Homework

Shell Sort

- Invented by Donald Shell in 1959.
- 1st algorithm to break the quadratic time barrier but few years later, a sub quadratic time bound was proven. Shellsort works by comparing elements that are distant rather than adjacent elements in an array.
- Shellsort improves on the efficiency of insertion sort by *quickly* shifting values to their destination
- Shellsort is also known as diminishing increment sort.
- The distance between comparisons decreases as the sorting algorithm runs until the last phase in which adjacent elements are compared

Advantage of Shellsort

- Advantage of Shellsort is that its only efficient for medium size lists. For bigger lists, the algorithm is not the best choice. Fastest of all O(N^2) sorting algorithms.
- 5 times faster than the bubble sort and a little over twice as fast as the insertion sort, its closest competitor.

Disadvantage of Shellsort

- Disadvantage of Shellsort is that it is a complex algorithm and its not nearly as efficient as the merge , heap, and quick sorts.

- The shell sort is still significantly slower than the merge , heap, and quick sorts , but its relatively simple algorithm makes it a good choice for sorting lists of less than 5000 items unless speed important. It's also an excellent choice for repetitive sorting of smaller lists.

Shell Sort example in Video

Shellsort Examples

Sort: 18 32 12 5 38 33 16 2

8 Numbers to be sorted, Shell's increment will be floor(n/2)

*** floor(8/2) ➔ floor(4) = 4**

increment 4: **1** **2** **3** **4** (visualize underlining)

18 32 12 5 38 33 16 2

Step **1**) Only look at **18** and **38** and sort in order ;
18 and **38** stays at its current position because they are in order.

Step **2**) Only look at **32** and **33** and sort in order ;
32 and **33** stays at its current position because they are in order.

Shellsort Examples

Sort: 18 32 12 5 38 33 16 2

8 Numbers to be sorted, Shell's increment will be floor(n/2)

*** floor(8/2) ➔ floor(4) = 4**

increment 4: **1** **2** **3** **4** (visualize underlining)

18 32 12 5 38 33 16 2

Step **3**) Only look at **12** and **16** and sort in order ;
12 and **16** stays at its current position because they are in order.

Step **4**) Only look at **5** and **2** and sort in order ;
2 and **5** need to be switched to be in order.

Shellsort Examples (con't)

Sort: 18 32 12 5 38 33 16 2

Resulting numbers after increment 4 pass:

18 32 12 *2* 38 33 16 *5*

*** floor(4/2) ➔ floor(2) = 2**

increment 2: **1 2**

18 32 12 2 38 33 16 5

Step **1**) Look at **18**, **12**, **38**, **16** and sort them in their appropriate location:

12 38 16 2 18 33 38 5

Step **2**) Look at **32**, **2**, **33**, **5** and sort them in their appropriate location:

12 2 16 5 18 32 38 33

Shellsort Examples (con't)

Sort: 18 32 12 5 38 33 16 2

*** floor(2/2) ➔ floor(1) = 1**

increment 1: 1

12	**2**	**16**	**5**	**18**	**32**	**38**	**33**
2	**5**	**12**	**16**	**18**	**32**	**33**	**38**

The last increment or phase of Shellsort is basically an Insertion Sort algorithm.

Shell Sort Algorithm

Step 1 − Initialize the value of h

Step 2 − Divide the list into smaller sub-list of equal interval h

Step 3 − Sort these sub-lists using insertion sort

Step 3 − Repeat until complete list is sorted

Outline

Sorting Algorithms

1- Quick Sort

2-Shell Sort

3- Homework

Assignment

What is the time complexity of shell sort? Analyze the worst case and best case with complete explanation.


وقل رب زدني علما
UNIVERSITY OF INFORMATION TECHNOLOGY AND COMMUNICATIONS


جامعة تكنولوجيا المعلومات والاتصالات
University of Information Technology & Communications
2014
Businesses Informatics College
كلية معلوماتية الاعمال

THANK YOU