CS122 Algorithms and Data Structures

MW 11:00 am - 12:15 pm, MSEC 101

Instructor: Xiao Qin

Lecture 21: Sorting (3)

1

Quicksort Algorithm

Given an array of *n* elements (e.g., integers):

- If array only contains one element, return
- Else
  - pick one element to use as *pivot.*
  - Partition elements into two sub-arrays:
    - Elements less than or equal to pivot
    - Elements greater than pivot
  - Quicksort two sub-arrays
  - Return results

2

Example

We are given array of n integers to sort:

40	20	10	80	60	50	7	30	100

3

Pick Pivot Element

There are a number of ways to pick the pivot element. In this example, we will use the first element in the array:

40	20	10	80	60	50	7	30	100

4

Partitioning Array

Given a pivot, partition the elements of the array such that the resulting array consists of:

1. One sub-array that contains elements >= pivot
2. Another sub-array that contains elements < pivot

The sub-arrays are stored in the original data array.

Partitioning loops through, swapping elements below/above pivot.

5

Partition Result

7	20	10	30	40	50	60	80	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

<= data[pivot] | > data[pivot]

6

Recursion: Quicksort Sub-arrays

7	20	10	30	40	50	60	80	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

<= data[pivot]   > data[pivot]

7

Quicksort Analysis

- Assume that keys are random, uniformly distributed.
- What is best case running time?
  - Recursion:
    1. Partition splits array in two sub-arrays of size n/2
    2. Quicksort each sub-array
  - Depth of recursion tree? $O(\log_2 n)$
  - Number of accesses in partition? O(n)

8

Quicksort Analysis

- Assume that keys are random, uniformly distributed.
- Best case running time: $O(n \log_2 n)$
- Worst case running time?

9

Quicksort: Worst Case

- Assume first element is chosen as pivot.
- Assume we get array that is already in order:

pivot_index = 0

2	4	10	12	13	50	57	63	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

too_big_index   too_small_index

10

pivot_index = 0

2	4	10	12	13	50	57	63	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

<= data[pivot]   > data[pivot]

11

Quicksort Analysis

- Assume that keys are random, uniformly distributed.
- Best case running time: $O(n \log_2 n)$
- Worst case running time?
  - Recursion:
    1. Partition splits array in two sub-arrays:
       - one sub-array of size 0
       - the other sub-array of size n-1
    2. Quicksort each sub-array
  - Depth of recursion tree? O(n)
  - Number of accesses per partition? O(n)

12

Quicksort Analysis

- Assume that keys are random, uniformly distributed.
- Best case running time: $O(n \log_2 n)$
- Worst case running time: $O(n^2)$!!!
- What can we do to avoid worst case?

13

Merge Sort

- Problem: Given *n* elements, sort elements into non-decreasing order
- Apply divide-and-conquer to sorting problem
  - If n=1 terminate (every one-element list is already sorted)
  - If n>1, partition elements into two sub-arrays; sort each; combine into a single sorted array
- How do we partition?

14

Partitioning

- Let's try to achieve balanced partitioning
- A gets *n/2* elements, B gets rest half
- Sort A and B recursively
- Combine sorted A and B using a process called *merge*, which combines two sorted lists into one
  - How?

15

Partitioning (cont.)

```
mergesort(data)
    if data have at least two elements then
        mergesort(left half of data);
        mergesort(right half of data);
        merge(both halves into a sorted list);
    endif
```

16

Evaluation

- Recurrence equation:
- Assume n is a power of 2

$$T(n) = \begin{cases} c_1 & \text{if } n=1 \\ 2T(n/2) + c_2 n & \text{if } n>1,\ n=2^k \end{cases}$$

17