VE281

Data Structures and Algorithms

Quick Sort

Outline

- Quick Sort
- Comparison Sort Summary

Quick Sort

Algorithm

Another divide-and-conquer approach to sort

- Choose an array element as **pivot**.
- Put all elements < pivot to the left of pivot.
- Put all elements ≥ pivot to the right of pivot.
- Move pivot to its correct place in the array.
- Sort left and right subarrays recursively (not including pivot).

`partition()`

```
void quicksort(int *a, int left,
  int right) {
    int pivotat; // index of the pivot
    if(left >= right) return;
    pivotat = partition(a, left, right);
    quicksort(a, left, pivotat-1);
    quicksort(a, pivotat+1, right);
}
```

Choice of Pivot

- If your input is random, you can choose the **first** element.
  - But this is very bad for presorted input.

- A better strategy: **randomly** pick an element from the array as pivot.
  - **Claim**: **for any input**, the average running time is $O(n \log n)$.
    - **Note**: average is over random choice of pivots made by the algorithm, **not** on the input.

Partitioning the Array

- Once pivot is chosen, swap pivot to the beginning of the array.
- When another array B is available, scan original array A from left to right.
  - Put elements < pivot at the left end of B.
  - Put elements ≥ pivot at the right end of B.
  - The pivot is put at the remaining position of B.
  - Copy B back to A.

A	6	2	8	5	11	10	4	1	9	7	3

B	2	5	4	1	3	6	7	9	10	11	8

In-Place Partitioning the Array

1. Once pivot is chosen, swap pivot to the beginning of the array.
2. Start counters **`i=1`** and **`j=N-1`**.
3. Increment **`i`** until we find element **`A[i]>=pivot`**.
   - **`A[i]`** is the leftmost item $\geq$ pivot.
4. Decrement **`j`** until we find element **`A[j]<pivot`**.
   - **`A[j]`** is the rightmost item $<$ pivot.
5. If **`i<j`**, swap **`A[i]`** with **`A[j]`**. Go back to step 3.
6. Otherwise, swap the first element (pivot) with **`A[j]`**.

In-Place Partitioning the Array

Example

`i` `j`

A	6	2	8	5	11	10	4	1	9	7	3
A	6	2	3	5	11	10	4	1	9	7	8
A	6	2	3	5	1	10	4	11	9	7	8
A	6	2	3	5	1	4	10	11	9	7	8

- Now, **`j < i`**, swap the first element (pivot) with **`A[j]`**.

A	4	2	3	5	1	6	10	11	9	7	8

In-Place Partitioning the Array

Time Complexity

1. Once pivot is chosen, swap pivot to the beginning of the array.
2. Start counters **`i=1`** and **`j=N-1`**.
3. Increment **`i`** until we find element **`A[i]>=pivot`**.
4. Decrement **`j`** until we find element **`A[j]<pivot`**.
5. If **`i<j`**, swap **`A[i]`** with **`A[j]`**. Go back to step 3.
6. Otherwise, swap the first element (pivot) with **`A[j]`**.

- Scan the entire array no more than twice.
- Time complexity is $O(N)$, where N is the size of the array.

Quick Sort

Time Complexity

```
void quicksort(int *a, int left,
  int right) {
    int pivotat; // index of the pivot
    if(left >= right) return;
    pivotat = partition(a, left, right);
    quicksort(a, left, pivotat-1);
    quicksort(a, pivotat+1, right);
}
```

$O(N)$

$T(LeftSz)$

$T(RightSz)$

- Let $T(N)$ be the time needed to sort N elements.
  - $T(0) = c$, where c is a constant.
- Recursive relation:

$$T(N) = T(LeftSz) + T(RightSz) + O(N)$$

  - $LeftSz + RightSz = N - 1$

Quick Sort

Worst Case Time Complexity

- Recursive relation:

$$T(N) = T(LeftSz) + T(RightSz) + O(N)$$

- Worst case happens when each time the pivot is the smallest item or the largest item
  - $T(N) = T(N-1) + T(0) + O(N)$

$$\begin{aligned} &\le T(N-1) + T(0) + dN \\ &\le T(N-2) + 2T(0) + d(N-1) + dN \\ &\quad \dots \\ &\le T(0) + NT(0) + d + 2d + \cdots + d(N-1) + dN \\ &= O(N^2) \end{aligned}$$

Quick Sort

Best Case Time Complexity

- Recursive realtaion:

$$T(N) = T(LeftSz) + T(RightSz) + O(N)$$

- Best case happens when each time the pivot divides the array into two equal-sized ones.
  - $T(N) = T((N-1)/2) + T((N-1)/2) + O(N)$
  - The recursive relation is similar to that of merge sort.
  - $T(N) = O(N \log N)$

Quick Sort

Average Case Time Complexity

- Average case time complexity of quick sort can be proved to be $O(N \log N)$.
  - Assume **randomly** pick an element from the array as pivot.
  - **Note**: average is over random choice of pivots made by the algorithm, **not** on the input.
  - The claim holds for any input.

Quick Sort

Other Characteristics

- In-place?
  - In-place partitioning.
  - Worst case needs $O(N)$ stack space.
  - Average case needs $O(\log N)$ stack space.
    - “Weekly” in-place.

- Not stable.

Quick Sort

Summary

- Like merge sort, quick sort is a divide-and-conquer algorithm.

- Merge sort: easy division, complex combination.
- Quick sort: complex division (partition with pivot step), easy combination.

- Insertion sort is faster than quick sort for small arrays.
  - Terminate quick sort when array size is below a threshold. Do insertion sort on subarrays.

Outline

- Quick Sort
- Comparison Sort Summary

Comparison Sorts
Summary

	Worst Case Time	Average Case Time	In Place	Stable
Insertion	$O(N^2)$	$O(N^2)$	Yes	Yes
Selection	$O(N^2)$	$O(N^2)$	Yes	No
Bubble	$O(N^2)$	$O(N^2)$	Yes	Yes
Merge Sort	$O(N \log N)$	$O(N \log N)$	No	Yes
Quick Sort	$O(N^2)$	$O(N \log N)$	Weakly	No

Comparison Sorts

Worst Case Time Complexity

- For comparison sort, is $O(N \log N)$ the best we can do in the worst case?

- Theorem: A sorting algorithm that is based on pairwise comparisons must use $\Omega(N \log N)$ operations to sort in the worst case.