ITEC2620 Introduction to Data Structures

Lecture 5a

Recursive Sorting Algorithms

Overview

- Previous sorting algorithms were **O**(n^2) on average
  - For 1 million records, that's 1 trillion operations – slow!
- What if we could get **O**(nℓogn)
  - For 1 million records, that's 20 million operations
  - 50,000 times faster!

O(nℓogn) Sorting

- Where have we seen **O**(ℓogn) before?
  - Binary search
- What is the difference between an **O**(n) algorithm and a **O**(ℓogn) algorithm?
  - **O**(ℓogn) achieves leverage
  - What is leverage?

Achieving Leverage I

- Linear search
  - Each unit of effort achieves one unit of work
    - First compare eliminates one location
    - Second compare eliminates one location
    - Third compare eliminates one location
    - Fourth compare eliminates one location

Achieving Leverage II

- Binary search
  - Each unit of effort achieves twice as many units of work
    - ➢First compare eliminates one location
    - ➢Second compare eliminates two locations
    - ➢Third compare eliminates four locations
    - ➢Fourth compare eliminates eight locations

Achieving Leverage III

- How is leverage achieved?
  - Divide and conquer
- Each division allows leverage
  - Search the middle (one division)
  - Search the middle of each half (two divisions)
  - Search the middle of each quarter (four divisions)

Recursive Sorting

- Split the elements into smaller divisions
- Partially sort each division (using the same algorithm)
- Use/trust recursion to put everything back together
- Each divisions allows leverage
  - Cannot achieve leverage with a simple algorithm that operates on the whole set

Quicksort Algorithm I

- Pick an element and partially sort it
  - Move all larger elements to one side, and all smaller elements to the other
    - ➢Create two divisions
  - Effort
    - ➢Compare all (remaining) elements
  - Work
    - ➢Get one element into final position

Quicksort Algorithm II

- Pick and partially sort an element in each division (2)
  - Move all elements as before (recursion)
    - ➢Create four divisions
  - Effort
    - ➢Compare all (remaining) elements
  - Work
    - ➢Get two elements into final position

Quicksort Algorithm III

- Pick and partially sort an element in each division (4)
  - Move all elements as before (recursion)
    - ➢Create eight divisions
  - Effort
    - ➢Compare all (remaining) elements
  - Work
    - ➢Get four elements into final position

Quicksort Algorithm IV

- Each division is being (partially) sorted with the same algorithm
  - This is the recursive sub-case
- What's the base case?
  - What is a trivial sorting problem?
    - ➢Sort 1 element
    - ➢Sort 0 elements

Quicksort Algorithm V

- Effort to sort each element is cut in half with each level of recursion
  - Equivalently, we get twice as much work done for our effort
- How many times can we cut something in half?
  - **O**($\log$n)

Quicksort Pseudocode I

```
public static void quicksort (int[] ar, int left, int right)
{
    // base case – 0 or 1 elements to sort
    if (left+1 >= right)
        return;

    // pick an element (in this division)
    int pivot = something;
```

Quicksort Pseudocode II

```
    // the partiallySort method moves all larger elements
    to one side and all smaller elements to the other
    pivot = partiallySort(ar, pivot, left, right)

    // continue recursively with the two new divisions
    quicksort(ar, left, pivot-1);
    quicksort(ar, pivot+1, right);
}
```

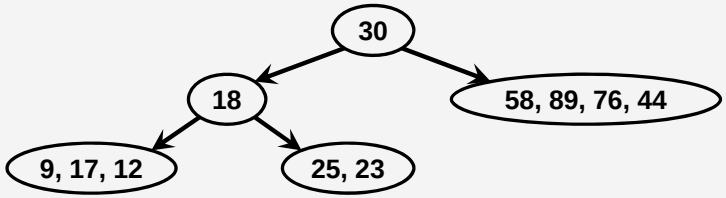
Quicksort Visualization I

76	58	25	18	12	30	44	17	89	9	23

- Pick a pivot (30)
- Partially sort it
  - Smaller on left, larger on right

9	17	25	18	12	23	30	58	89	76	44

Quicksort Visualization II



Partiallysort Pseudocode I

```
public static int partiallySort
        (int[] ar, int pivot, int left, int right)
{
        int pivotValue = ar[pivot];

        // move the pivot to a safe place
        swap (ar, pivot, right);
```

Partiallysort Pseudocode II

```
        // make two divisions
        while (moreToSwap)
        {
                int larger = findLargerOnLeft();
                int smaller = findSmallerOnRight();
                swap (ar, larger, smaller);
        }
        swap (ar, right, final);
        return final;
}
```

Partiallysort Visualization

76	58	25	18	12	30	44	17	89	9	23
76	58	25	18	12	23	44	17	89	9	30
9	58	25	18	12	23	44	17	89	76	30
9	17	25	18	12	23	44	58	89	76	30
9	17	25	18	12	23	30	58	89	76	44

Mergesort Algorithm I

- Divide what you have to do into two halves/divisions
- Sort each half/division
- Merge the two halves/divisions into a fully sorted set

Mergesort Algorithm II

- Each half/division will be sorted by the same algorithm as the overall set
  - This is the recursive sub-case
- What's the base case?
  - What is a trivial sorting problem?
    - Sort 1 element
    - Sort 0 elements

Mergesort Algorithm III

- Each upward merge sorts twice as much
- How many times can we cut something in half?
  - **O**($\log$n)

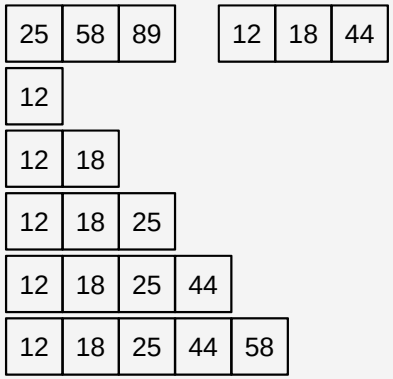
Mergesort Pseudocode

```
public static void mergesort (int[] ar, int left, int right)
{
    if (left+1 >= right)
        return;

    mergesort (ar, left, middle);
    mergesort (ar, middle+1, right);

    merge(ar, left, middle, right);
}
```

Merge Pseudocode

```
public static void merge (int[] ar, int left, int m, int right)
{
    for (int i = 0; i < ar.length; i++)
    {
        if (leftSideSmaller)
            ar[i] = temp[leftSide];
        else
            ar[i] = temp[rightSide];
    }
}
```

Merge Visualization



Readings and Assignments

- Suggested Readings from Shaffer (third edition)
  - 7.4, 7.5