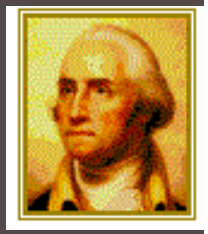


THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON DC

DIVIDE AND CONQUER

PART II

CLOSEST PAIR OF POINTS
MEDIAN FINDING
SELECTION ALGORITHMS

Design and Analysis of Algorithms

LOGISTICS

- Instructor

  Prof. Amrinder Arora

  amrinder@gwu.edu

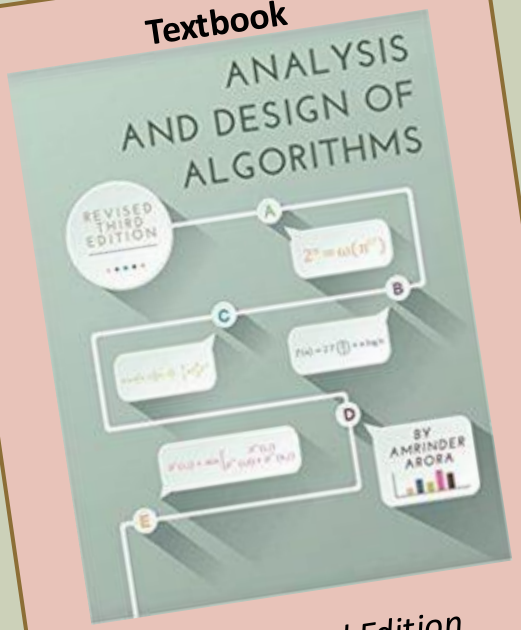
  Please copy TA on emails

  Please feel free to call as well

  ☺

- Available for study sessions
  Science and Engineering Hall
  GWU

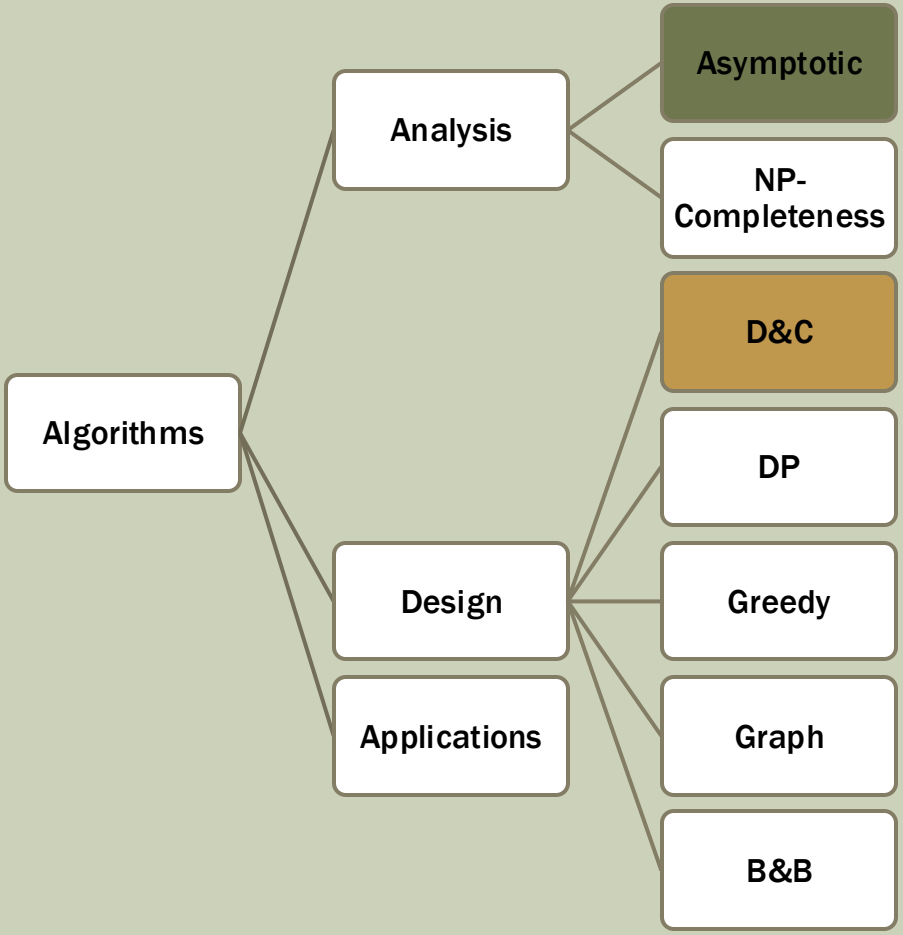
Textbook



Revised Third Edition
Cognella Academic Publisher
www.cognella.com

ISBN: **978-1793549952**

WHERE WE ARE


Algorithms
Analysis
Asymptotic
NP-Completeness
Design
D&C
DP
Greedy
Applications
Graph
B&B

DIVIDE AND CONQUER

- A technique to solve complex problems by breaking into smaller instances of the problem and combining the results
  - Recursive methodology – Smaller instances of the same *type* of problem
- Typically used with:
  - Principle of Mathematical Induction – For proving correctness
  - Recurrence relation solving – For computing time (and/or space) complexity

MERGESORT

- Generalization step: Given an array *and indexes i and j (start and end)* to sort *that portion* of it
- Algorithm MergeSort (input: A,i,j) {

```
    if input size is small, solve differently and return
    int k=(i+j)/2
    MergeSort(A,i,k)
    MergeSort(A,k+1,j)
    Merge(A,i,k,k+1,j)
}
```

- T(n) = 2T(n/2) + n
- T(n) = O(n log n)

[Discussed in previous lectures.]

QUICKSORT

- quicksort(A,p,r)

```
if (p < r) {
        q = partition (A,p,r)
        quicksort(A,p,q-1)
        quicksort(A,q+1,r)
}
```

- Time complexity = ?

[Discussed in previous lectures.]

MEDIAN FINDING

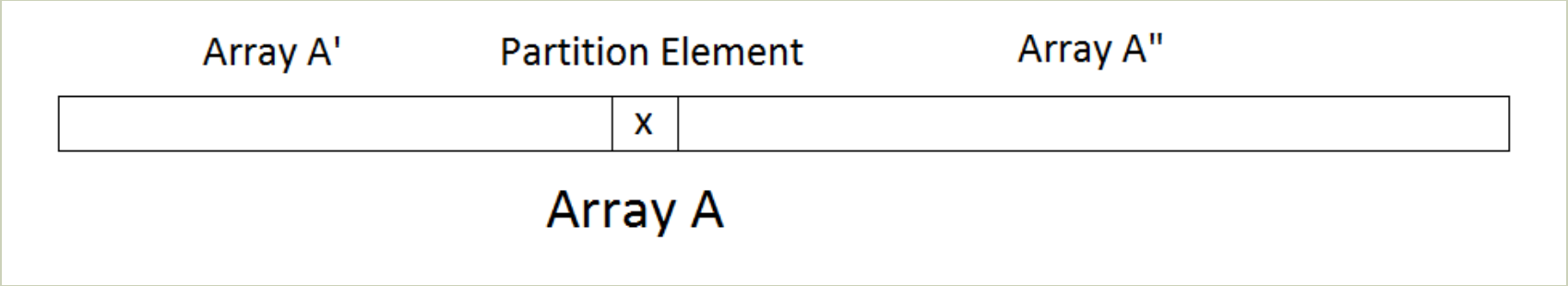
- Given an array of unsorted numbers, how to find the median?
- Option 1: Sort and return a[n/2]
- That takes O(n log n) time. Is there a more efficient algorithm?

MEDIAN FINDING

- Step 1: Generalize the problem
- Selection (A, k): Returns *k*-th smallest element in given array. For example:
  - Selection (A,1) = Smallest
  - Selection(A,n) = Largest // n is the array size.
  - Selection(A,n/2) = Median
- Selection algorithms: http://en.wikipedia.org/wiki/Selection_algorithm
- Also known as order statistic.
- Generalization of largest, smallest, median, etc. problems.
- This generalization step is **very important** for recursive calls!

SELECTION – GENERAL TEMPLATE

```
Selection (A,k)
Partition the array A
```




```
Suppose, the partition element lands at location k'
If (k == k') {
 return x                        // Great, we really got lucky.
}
If (k < k') {
 return Selection (A',k)         // Recursive call.  Discard A"
}
If (k > k') {
 return Selection (A",k-k')      // Recursive call.  Discard A'
}
```

EXAMPLE PARTITION

- Selection(A, 5)

- A = [3, 5, 90, 4, 8, 1, 6, 11, 9, 10, 100]
- Random partition element = 6
- [A1 P A2]
- [3, 5, 4, 1, 6, 100, 10, 9, 11, 8, 90]
- [Location of element 6] = 5 = k' = k
- Return 6! (We are lucky)

EXAMPLE PARTITION

- Selection(A, 8)

- A = [3, 5, 90, 4, 8, 1, 6, 11, 9, 10, 100]
- Random partition element = 6
- [A1 P A2]
- [3, 5, 4, 1, 6, 100, 10, 9, 11, 8, 90]
- [Location of element 6] = 5 = k' < (k = 8)
- Therefore, the Selection (A,8), must be in the Array A2.
- Selection (A,8) = Selection (A2, 8 – 5)
  - Now need to make a (recursive) call for Selection (A2,3)
  - [Because 5 elements were eliminated in the highlighted portion of the array]

EXAMPLE PARTITION

- Selection(A, 2)

- A = [3, 5, 90, 4, 8, 1, 6, 11, 9, 10, 100]
- Random partition element = 6
- [A1 P A2]
- [3, 5, 4, 1, 6, 100, 10, 9, 11, 8, 90]
- [Location of element 6] = 5 = k' > (k = 2)
- Therefore, the Selection (A,2), must be in the Array A1.
- Selection (A,8) = Selection (A1, 2)
  - [Because elements were eliminated from the "right" side of the array]

SELECTION – TWO APPROACHES

- The main complication (as in case of Quicksort) is to find a good partition. If the partition is very uneven, the algorithm may not be efficient.
- We have two good choices:
  - Use probability to find a good partition by trying repeatedly
  - Find a good partition deterministically
- In the coming few slides, we explore both of these methods.

SELECTION – PROBABILISTIC METHOD

- Partition randomly.
- Define: A "good" partition is something that lands between the middle half, that is between ¼ and ¾ of the array.
- You can find a "good" partition with probability ½
- Expected number of times that you have to do a partition until you get a "good" partition = 2
- Suppose the good partition "lands" at location k', where n/4 <= k' <= 3n/4
- Depending upon value of k and k', we make the appropriate recursive call, as specified in general template.
- After partition, we remove at least 25% of the array.
- T(n) <= 2n + T(0.75 n) // 2 reflects the expected # of times that we have to do the partition
- T(n) <= cn // Using substitution method
- Therefore, T(n) = O(n)

SOLVING RECURRENCE RELATIONS

- $T(n) = 2n + T(3n/4)$
- $a = 1$
- $b = 4/3$
- $f(n) = 2n$
- $\log_b(a) = 0$
- $n^0 = 1$
- f(n) term dominates
- $T(n) = f(n) = n$

SELECTION – QUICKSELECT ALGORITHM

- Divide the array of size n into groups of 5 (or 7)
- Sort the small groups
- Find the median of the medians
- Partition the array on that median

X X X X X ..... x x x x x
X X X X X ..... x x x x x
X X X X X ..... x x x x x
x x x x x ..... x x x x x
x x x x x ..... x x x x x

QUICKSELECT (CONT.)

- Say x = the median of median
- Then at least 30% of elements are <= x
- At least 30% of elements are >= x
- We use this x to be the partition element
- Suppose x "lands" at location k', where 3n/10 <= k' <= 7n/10
- Depending upon value of k and k', we make the appropriate recursive call, as specified in general template.
- Depending upon the value of k, either the left side of the partition or the right side will be discarded. In either case, we eliminate at least 30% of data.
- We note that there are 2 recursive calls – 1st call to find the median of medians, and 2nd call after removing 25% of elements

QUICKSELECT – ANALYSIS

- $T(n) = cn + T(n/5) + T(7n/10)$

- Proof that T(n) is linear, that is O(n), by substitution method (Using Principle of Mathematical Induction)
  We claim that $T(n) <= 10\,cn$ for all values of $n < N$
  $T(N/5) <= 2cN$
  $T(7N/10) <= 7cN$
  → $T(N) <= cN + 2cN + 7cN = 10cN$
  → The inequality holds for N
  → By principle of mathematical induction, inequality holds for all values of n.
- Therefore, T(n) = O(n)

DIVIDE AND CONQUER – ANOTHER APPLICATION

- Given n points on the plane, find the closest pair of points.

- http://en.wikipedia.org/wiki/Closest_pair_of_points_problem
- Textbook section 4.7

CLOSEST PAIR OF POINTS

- Naïve algorithm requires $O(n^2)$ time – simply compute all distances and find the minimum.
- We are interested in a divide and conquer approach.
- We remember that there is a significant difference between $O(n^2)$ time algorithm and $O(n \log n)$ time algorithm. For example, latter can easily be run with a trillion points, the $O(n^2)$ algorithm, not so much.

CLOSEST PAIR OF POINTS (CONT.)

- Divide and Conquer approach:
  1. Split the set of points into two equal-sized subsets by finding the median of the x-dimensions of the points.
  2. Solve the problem recursively for subsets to the left and right subsets. Thus, there are two recursive calls.
  3. Find the minimal distance among the pair of points in which one point lies on the left of the dividing vertical and the second point lies to the right.
- $T(n) = cn + 2T(n/2) + f(n)$, where:
  - cn represents the time spent in Step 1 (linear time median finding)
  - $2T(n/2)$ is time spent in two recursive calls in Step 2.
  - f(n) is the time spent in step 3, that is, time to find the minimum distance among the pairs, where one point lies to the left of the dividing vertical, and the second point lies to the right.

CLOSEST PAIR OF POINTS (CONT.)

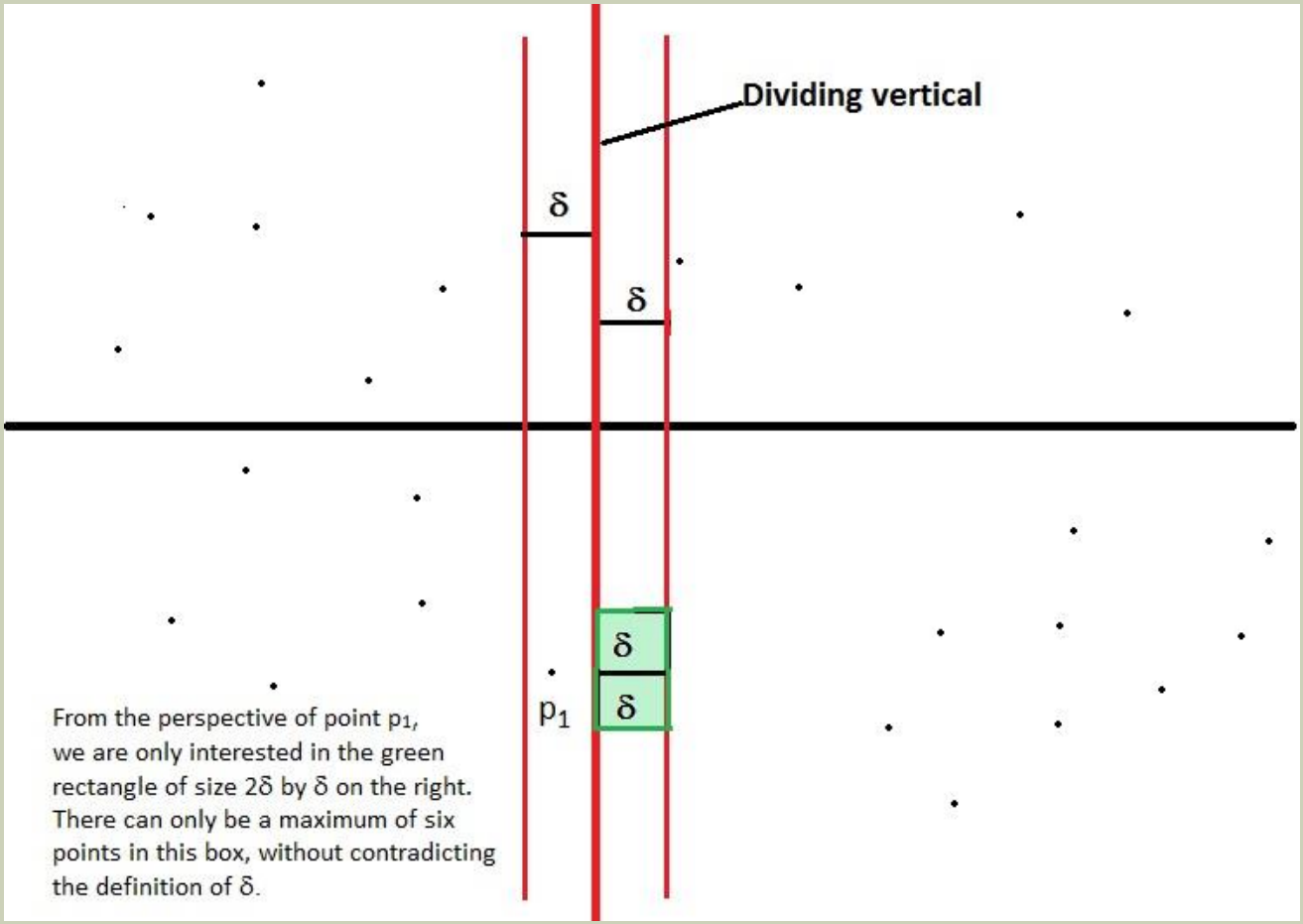
- $T(n) = 2T(n/2) + cn + f(n)$
  - If $f(n) = O(n^2)$, then, $T(n) = O(n^2)$
  - If $f(n) = O(n)$, then $T(n) = O(n \log n)$
- Main complication then is to bound f(n)
- If we can do f(n) in linear time, we will have an O(n log n) time algorithm.

- In the next couple of slides, we discuss indeed how this can be done in linear time.

CLOSEST PAIR OF POINTS (CONT.)

- We define:
  - δ_1 = Minimum distance found in the left side
  - δ_2 = Minimum distance found in the right side
  - δ = Minimum of δ_1 and δ_2
- Firstly, we are only interested in the points that are within a distance δ of the dividing vertical.
- For a point p_1 on the left side of the dividing vertical, we are only interested in:
  - Points that are on the right side of the vertical, and within a distance δ from the dividing vertical
  - Points that are within a vertical distance δ from the point p_1.

CLOSEST PAIR OF POINTS (CONT.)



CLOSEST PAIR OF POINTS (CONT.)

- Thus for a point p_1, there can only be 6 points of interest for that point.
- Even if there are n/2 points to the left of the dividing vertical and within a distance δ, still that only means 3n pairs to consider

- Therefore, f(n) = O(n)

- Therefore, T(n) = 2T(n/2) + O(n)
- Therefore, T(n) = O(n log n)

CLOSEST PAIR OF POINTS (CONT.)

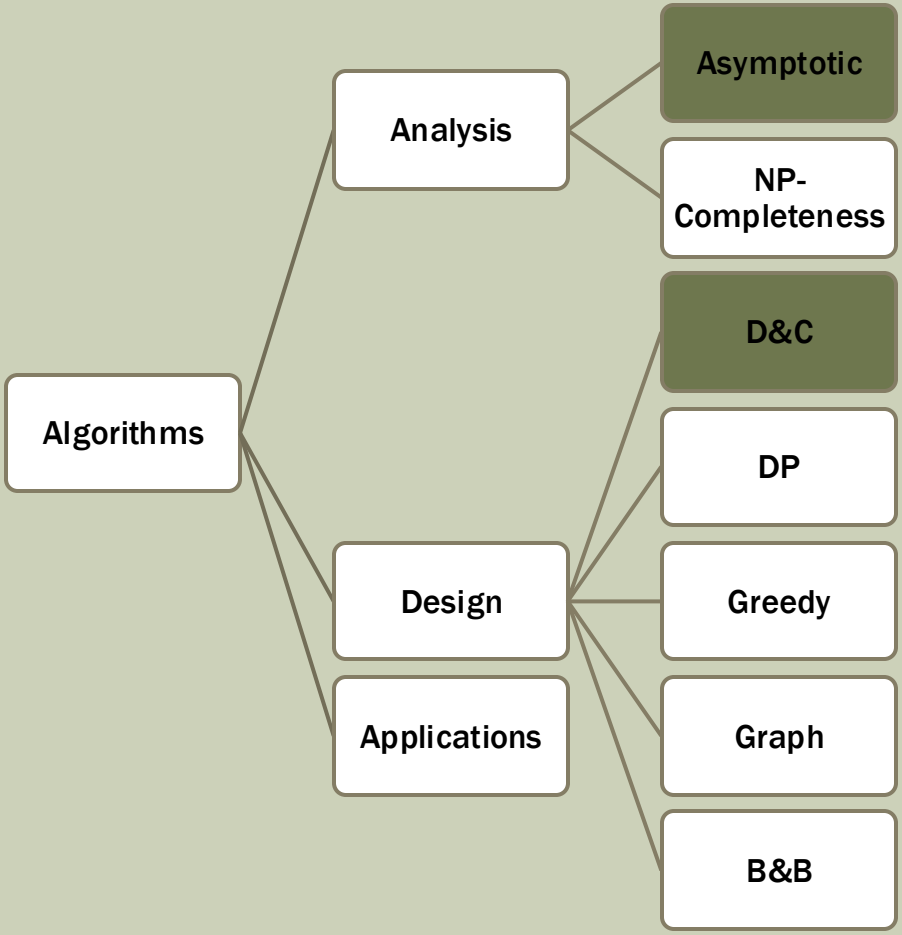
- Considering there are so few points in the middle vertical bar, if we can do the merging part in less than linear time, for example in O(n^0.75) time, that may be very interesting.
- Hypothetically, then T(n) = 2T(n/2) + O(n^0.75)
- Then, we can have T(n) = O(n) // Linear time

- Might that be possible? How can we get there? (Or if not, why not?)

SUMMARY AND WRAP UP

- D&C – An interesting technique for solving problems
- Three different ways to solve recurrence relations
  - Unfolding method
  - Substitution (guess and prove method)
  - Master Theorem – Another Method for solving Recurrence Relations
- QuickSelect – O(n) time algorithm for selecting k-th largest element in an unsorted array.
  - The constant is rather large, use this algorithm with caution
- Closest pair of points – An interesting O(n log n) time recursive algorithm for finding the closest pair of points.

WHERE WE ARE


Algorithms
Analysis
Asymptotic
NP-Completeness
Design
D&C
DP
Greedy
Graph
B&B
Applications

READING ASSIGNMENTS

- Strassen's algorithm for matrix multiplication (Textbook § 4.8)
- Greedy Algorithms ((Textbook § 5.1 – 5.4)
- Section 4.2 in http://compgeom.cs.uiuc.edu/~jeffe/teaching/algorithms/notes/04-greedy.pdf