Divide and Conquer

- A General principle of designing algorithms.
- It applies to many problems, particular those with a "parallel flavor".
- Algorithm has 3 steps:
  1. Divide the data into parts
  2. Find sub-solutions for each of the parts (usually) recursively
  3. Construct the final answer for the original problem from the sub-solutions
- Examples:
  - Binary Search, Merge Sort, Quicksort

Control Abstraction for Divide and Conquer

Algorithm DC(P)
{
  if small(P) then return S(P);
  else
  {
    divide P into smaller instances
    $P_1, \ldots, P_k, k > 1$;
    Apply DC to each of these subproblems;
    return combine(DC(P_1),...,DC(P_k));
  }
}

Recursive Binary Search

Algorithm RBS (a,i,l,x)
//Given an array $a[i:l]$ of elements in non-decreasing
//order, $1 \leq i \leq l$, determine whether x is present,
//and if so, return j such that $x = a[j]$; else return 0

{
  if ($i \leq l$) then
  {
    $mid := \lfloor \frac{(i+l)}{2} \rfloor$;
    if ($x = a[mid]$) then return mid;
    else if ($x < a[mid]$)
      then return $RBS(a, i, mid-1, x)$;
      else return $RBS(a, mid+1, l, x)$;
  }
  else
    return 0;
}

Iterative Binary Search

Algorithm $IBS(a, n, x)$
//$a[1:n]$
{
  $low := 1; high := n$;
  while ($low \leq high$) do
  {
    $mid := \lfloor \frac{low+high}{2} \rfloor$;
    if ($x < a[mid]$) then $high := mid - 1$;
    else if ($x > a[mid]$) then $low := mid + 1$;
    else return mid;
  }
  return 0;
}

An Example

-25, -12, -6, 0, 1, 2, 3, 5, 7, 11, 13, 17, 19, 23

x=23, the search sequence

low	high	mid
1	14	7
8	14	11
12	14	13
14	14	14
		found

x=-24, the search sequence

low	high	mid
1	14	7
1	6	3
1	2	1
2	2	2
2	1	
		not found

x=1, the search sequence

low	high	mid
1	14	7
1	6	3
4	6	5
		found

A Useful Application of Binary Search

Partition Problem

Given an arrangement S of non negative numbers $\{s_1, \ldots, s_n\}$ and an integer k. Partition S into k ranges so as to minimize the maximum sum over all ranges.

```
100 100 100 | 100 100 100 | 100 100 100
|---300-----|-----300-----|-----300----|


100 200 300 | 400 500 600 | 700 800 900
|---600-----|----1500-----|----2400----|


100 200 300 400 500 | 600 700 |  800 900
|-------1500--------|---1300--|---1700--|
```

Solution?

Finding Maximum and Minimum

To find the maximum and minimum of a set of n numbers

Algorithm Simple(a, n, max, min)
//Set max to maximum and min to
//minimum of $a[1:n]$
{
  $max := min := a[1]$
  for i:=2 to n do
  {
    if ($a[i] > max$) then $max := a[i]$;
    if ($a[i] < min$) then $min := a[i]$;
  }
}

How many comparisons are there?

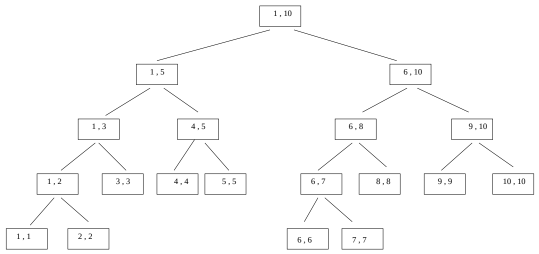
Algorithm RMaxMin(i, j, max, min)

{
  if ($i = j$) then $max := min := a[i]$;
  else if ($i = j - 1$) then
    if ($a[i] < a[j]$) then
    {
      $max := a[j]$; $min := a[i]$;
    }
    else
    {
      $max := a[i]$; $min := a[j]$;
    }
  else
  {
    $mid := \lfloor \frac{(i+j)}{2} \rfloor$;
    RMaxMin(i, mid, max, min);
    RMaxMin($mid + 1, j, max1, min1$);
    if ($max < max1$) then $max := max1$;
    if ($min > min1$) then $min := min1$:
  }
}

Number of comparisons?

Merge Sort

$A[1:10] =$
$\{310,285,179,652,351,423,861,254,450,520\}$



$$T(n) = \begin{cases} a & n = 1 \text{ and } a \text{ is a constant} \\ 2T(\frac{n}{2}) + cn & n > 1 \text{ and } c \text{ is a constant} \end{cases}$$

$T(n) = O(n \log n)$

Algorithm for Mergesort

Algorithm MergeSort($low, high$)
//$a[low : high]$ is a global array to be sorted
//small(P) is true if there is only 1 element to sort
{
  if ($low < high$)
  {
    $mid := \lfloor \frac{low+high}{2} \rfloor$;
    mergesort(low, mid);
    mergesort($mid + 1, high$);
    merge($low, mid, high$);
  }
}

Problem : Inversion Index

Given a list of n unique integers $\{a_1, a_2, \ldots, a_n\}$, where $1 \le a_i \le n$ for $1 \le i \le n$, count the number of inversions. a_i and a_j are inverted if and only if $i < j$ but $a_i > a_j$.

For example
The inversion index of
$\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ is 0.
The inversion index of
$\{1, 5, 4, 8, 10, 2, 6, 9, 12, 11, 3, 7\}$ is 22.
You are to give the fastest algorithm to solve the problem.

QuickSort

Algorithm QuickSort(p, q)
//Sort the elements $a[p]$,...,$a[q]$
//which reside in the global array
//$a[1...n]$ into ascending order;
//$a[n + 1]$ is considered to be defined
//and must be $\ge$ all the elements in $a[1 : n]$
{
  if ($p < q$) then
  {
    //divide p into 2 subproblems
    j:= Partition($a, p, q + 1$)
    //j is the position of the partitioning element
    //solve subproblems
    QuickSort($p, j - 1$);
    QuickSort($j + 1, q$);
  } //No need to combine solution
}

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
65	70	75	80	85	60	55	50	45	+ Infinity
65	45	75	80	85	60	55	50	70	+ Infinity
65	45	50	80	85	60	55	75	70	+ Infinity
65	45	50	55	85	60	80	75	70	+ Infinity
65	45	50	55	60	85	80	75	70	+ Infinity
60	45	50	55	65	85	80	75	70	+ Infinity

Strassen's Matrix Multiplication

$C = AB$

$$A = \left[\begin{array}{c:c} A_{11} & A_{12} \\ \hdashline A_{21} & A_{22} \end{array}\right] \qquad B = \left[\begin{array}{c:c} B_{11} & B_{12} \\ \hdashline B_{21} & B_{22} \end{array}\right] \qquad C = \left[\begin{array}{c:c} C_{11} & C_{12} \\ \hdashline C_{21} & C_{22} \end{array}\right]$$

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$
$C_{12} = A_{11}B_{12} + A_{12}B_{22}$
$C_{21} = A_{21}B_{11} + A_{22}B_{21}$
$C_{22} - A_{21}B_{12} + A_{22}B_{22}$

$$T(n) = \begin{cases} b & n \leq 2 \\ 8T(\frac{n}{2}) + Cn^2 & n > 2 \end{cases}$$

$$T(n) = O(n^3) + O(n^2)$$

With Some Manipulations

$p = (A_{11} + A_{22})(B_{11} + B_{22})$
$Q = (A_{21} + A_{22})B_{11}$
$R = A_{11}(B_{12} - B_{22})$
$S = A_{22}(B_{21} - B_{11})$
$T = (A_{11} + A_{12})B_{22}$
$U = (A_{21} - A_{11})(B_{11} + B_{12})$
$V = (A_{12} - A_{22})(B_{21} + B_{22})$

$$\left.\begin{array}{l} C_{11} = P + S - T + V \\ C_{12} = R + T \\ C_{21} = Q + S \\ C_{22} = P + R - Q + U \end{array}\right\} \text{7 X and 18 +/-}$$

$$T(n) = \begin{cases} b & n \leq 2 \\ 7T(\frac{n}{2}) + an^2 & n > 2 \end{cases}$$

Rank

Rank of an element is its position in ascending key order

[2,6,7,8,10,15,18,20,25,30,35,40]

rank(2) = 0
rank(15)= 5
rank(20)= 7

Selection Problem

- Given n unsorted elements, determine the k'th smallest element. i.e. determine the element whose rank is $k-1$.
- Applications
  - Median score on a test.
    - $k = \lceil \frac{n}{2} \rceil$
  - Median salary of Computer Scientists
  - Identify people whose salary is in the bottom 10%. First find salary at the 10% rank.

Selection By Sorting

- Sort the n elements
- Pick up the element with desired rank
- $O(n \log n)$ time

Divide-And-Conquer Selection

- Small instance has $n \leq 1$. Selection is easy.
- When $n > 1$, select a pivot element from out of the n elements.
- Partition the n elements into 3 groups: left, middle, and right as is done in quicksort.
- The rank of the pivot is the location of the pivot following the partitioning.
- If $k-1 = rank(pivot), pivot$ is the desired element.
- if $k-1 < rank(pivot)$, determine the k'th smallest element in left.
- if $k-1 > rank(pivot)$, determine the $(k - rank(pivot) - 1)$'th smallest element in right.

Example

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

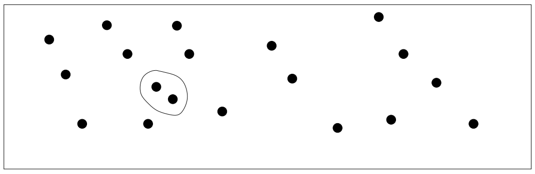
- Let the pivot be $a = 6$.
- $rank(pivot) = 5$. So $pivot$ is the 6'th smallest element
- If $k = 6$ $(k-1 = rank(pivot))$, $pivot$ is the element we seek
- If $k < 6$ $(k-1 < rank(pivot))$, find the k'th smallest element in left partition
- If $k > 6$ $(k-1 > rank(pivot))$, find $(k - rank(pivot) - 1)$'th smallest element in right partition

Time Complexity

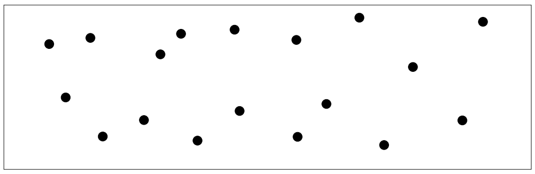
- Worst case arises when the partition to be searched always has all but the pivot
  - $O(n^2)$
- Expected performance is $O(n)$
- Worst case becomes $O(n)$ when the pivot is chosen carefully
  - Partition into $\frac{n}{9}$ groups with 9 elements each (last group may have a few more)
  - Find the median element in each group
  - pivot is the median of the group medians
  - This median is found using select recursively

Closest Pair of Points

- Given n points, find the pair that are closest.



Applications



- We plan to drill holes in a metal sheet
- If the holes are too close, the sheet will tear during drilling
- Verify that no two holes are closer than a threshold distance (e.g. holes are at least 1 inch apart)

Air Traffic Control

- 3D-Locations of airplanes flying in the neighborhood of a busy airport ar known
- Want to be sure that no two planes get closer than a given threshold distance

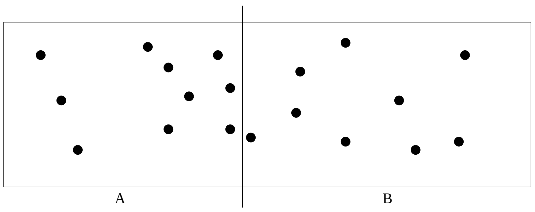
Simple Solution

- For each of the $\frac{n(n-1)}{2}$ pairs of points, determine the distance between the points in the pair
- Determine the pair with the minimum distance
- $O(n^2)$ time

Divide-And-Conquer Solution

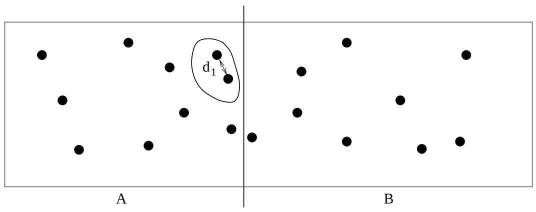
- When n is small, use simple solution
- When n is large
  - Divide the point set into two roughly equal parts A and B
  - Determine the closest pair of points in A
  - Determine the closest pair of points in B
  - Determine the closest pair of points such that one point is in A and the other is in B
  - From the three closest pairs computed, select the one with the least distance

Example



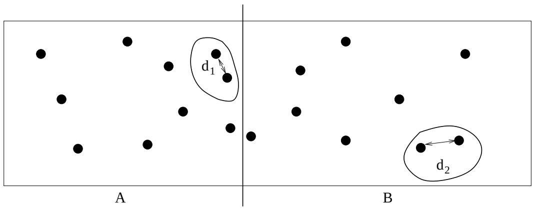
- Divide so that the points in A have x-coordinate $\leq$ that of points in B

Example



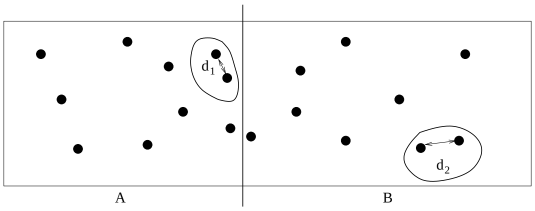
- Find closest pair in A.
- Let d_1 be the distance between the points in this pair.

Example



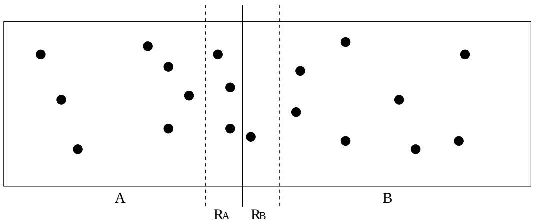
- Find the closest pair in B.
- Let d_2 be the distance between the points in this pair.

Example



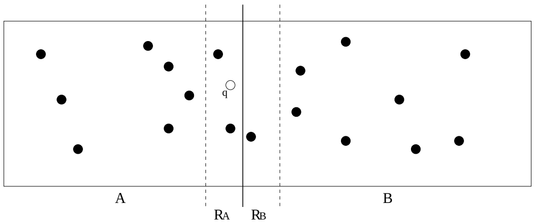
- Let $d = \min\{d_1, d_2\}$.
- Is there a pair with one point in A, the other in B and the distance $\leq d$?

Example



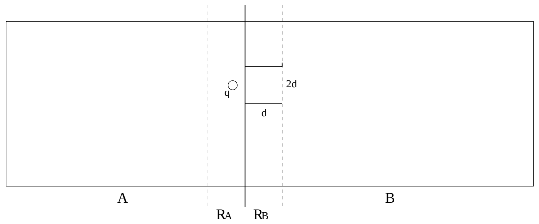
- Candidates lie within d of the dividing line.
- Call these regions R_A and R_B respectively.

Example



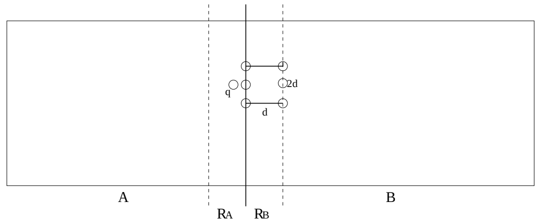
- Let q be a point in R_A.
- q needs to be paired only with those points in R_B that are within d of q.

Example



- Points that are to be paired with q are in a $d \times 2d$ rectangle of R_B (comparing region q).
- Points in this rectangle are at least d apart.

Example



- So the comparing regional of q has at most 6 points.
- So number of pairs to check is $\leq 6|R_A| = O(n)$.

Time Complexity

- Create a sorted by x-coordinate list of points - $O(n \log n)$ time.
- Create a sorted by y-coordinate list of points - $O(n \log n)$ time.
- Using these two lists, the required pairs of points from R_A and R_B can be constructed in $O(n)$ time.
- Let $n < 4$ define a small instance.

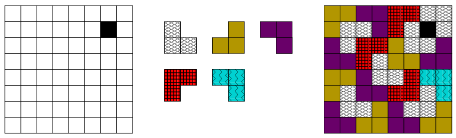
Time Complexity

- Let $T(n)$ be the time to find the closest pair (excluding the time to create the two sorted lists).
- $T(n) = c$ when $n < 4$ where c is a constant.
- When $n \geq 4$, $T(n) = T(\lceil (n/2) \rceil) + T(\lfloor (n/2) \rfloor) + dn$, where d is a constant.
- Using Master's Theorem, $T(n) = O(n \log n)$.

Exercise

Tiling Using Triominoes

Given a 2^k by 2^k chessboard where $k \geq 1$ with a defective square, you are to tile the chessboard using triominoes. Note that the defective square can occur anywhere in chessboard. A 8×8 chessboard with a defective square is given in Figure (a). A triominoes is an "L"-shaped tile formed by three adjacent squares of a chessboard. See Figure (b) for examples of triominoes. Figure (c) shows a tiling of the chessboard with triominoes that covers every square of the board except the defective one.



(a) A defective chessboard (b) Triominoes (c) Tiling of defective chessboard using tri0minoes

1. Devise an algorithm that solves this problem.
2. If no pair of triominoes that share an edge have the same color, how many colors of triominoes do you need.

Exercise

Maximum Subsequence Sum

The *Maximum Subsequence Sum* problem is defined as follows. Given an array $A[1 \ldots n]$ of integers, find values of i and j with $1 \leq i \leq j \leq n$ such that

$$\sum_{k=i}^{j} A[k]$$

is maximized.

1. Devise an algorithm to solve the problem in $O(n^3)$.
2. Devise an algorithm to solve the problem in $O(n^2)$.
3. Devise an divide-and-conquer algorithm for solving the maximum subsequence problem in time $O(n \log n)$.
4. Devise an algorithm to solve the problem in $O(n)$.

Exercise

Nearest to Zero Subsequence Sum

The *Nearest to Zero Subsequence Sum* problem is defined as follows. Given an array $A[1 \ldots n]$ of integers, find values of i and j with $1 \leq i < j \leq n$ such that

$$\sum_{k=i}^{j} A[k]$$

is as near to zero as possible.

1. Devise an algorithm to solve the problem in $O(n^3)$.
2. Devise an algorithm to solve the problem in $O(n^2)$.
3. Devise an algorithm to solve the problem in $O(n \log n)$.