Quiz notes

Divide and Conquer 1

Paradigm:

To solve problem size n
- if n is small:
  - solve directly
- else:
  - divide -> into subproblems which are smaller instances of problem
  - conquer -> solve recursively, if subprob small enough solve directly
  - combine -> combine subprob solution to original

Example - Binary Search

```
Binary-Search (A, t):
  If A has less than 4 elements:
      Perform sequential search on A in order to find t
  else:
      Divide A into two equal subsequences
      Compare t with the last element of the first half of A
      Binary-Search(first half or second half of A, t)
```

5, 6 ... dir

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

Mergesort.

```
Merge-Sort (A):
  If A has less than 3 elements:
      Sort A using any algorithm
  else:
      Merge-Sort(first half of A)
      Merge-Sort(second half of A)
      Merge the two sorted halves together into a full sorted set
```

dir

divide + conquer

combine

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

What about divide into 3 sublists?

does not improve time eff of merge sort

```
Quick-Sort (A):
  If A has less than 4 elements:
      Sort A using any algorithm
  else:
      p = Pivot(A)
      q = Partition(A, p)
      Quick-Sort(A[1,...,q-1])
      Quick-Sort(A[q,...,n])
```

best = $\Theta(n \log n)$

worst = $\Theta(n^2)$

DnC 2

Max subarray

Algorithm

[[ ]]

Max-subarray(A):

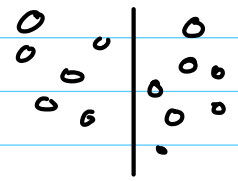
if |A| ≤ s:
    brute force to find max subarray

else:
    Divide to A[1...mid] and A[mid+1...n]
    maxsub1(A[1..mid])
    maxsub(A[mid+1....n])
    Max_Crossing-subarray(A) → middle of halving
    Return one of the three with greatest

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

DnC 3

Closest Pair



closest_pair(P):

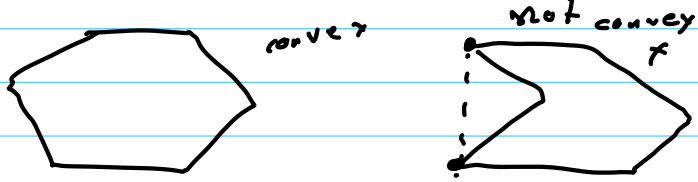
$O(n \log n)$

if |P| ≤ 4:
    find closest using brute force
else:
    Divide set into a LR half P_L and P_R
    δ_L = Closest_pair(P_L)
    δ_R = Closest_pair(P_R)
    δ = min(δ_L, δ_R)
    if there is a pair belongs to one side & d < δ:
        return d
    else:
        return δ

Dnc 4

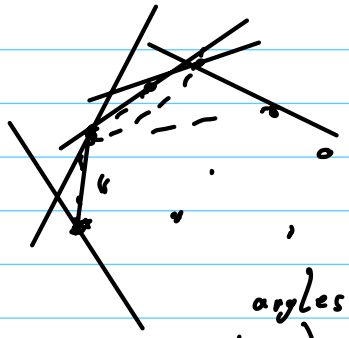
Convex Hull

Smallest convex polygon that includes all points
Conver poly: no points in polygon so that straight line between goes outside polygon



Brute force $O(n^3)$ -> $n \cdot (n-1) \cdot (n-2)$

1. Jarvis's march algo



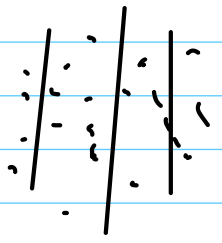
1. Find left most point $O(n)$
2. For each:

iterate h times {
- -> compute angles to remaining
- -> pick point with smallest
- -> include one more point to CH(Q)

angles
$\underbrace{(n-1)+(n-1)+(n-2)\ldots.}_{O(n^2)}$

h = no. of points
$O(n \times h)$
h because of points inside

What about DnC



1. Divide
2. Divide until 3 on each side
3. Join right most (left side) and left most (right)
4. Join points which have angles >180

Pair Sum

Given set S of n integers, target k, does S contain a pair of elems that sum to k?

Algo

-> sort S in ascending order $\quad S = \{s_1, s_2 \dots s_n\}$
$\quad\quad s_1 \leq s_2 \leq s_3 \cdots$
-> compute $t = s_1 + s_n$
$t = k$, found
$t < k$, remove s_1, $t = s_2 + s_n$
$t > k$, remove s_n, $t = s_1 + s_{n-1}$

Pseudocode

```
Pairsum (S, k):
 sort S                                  -----> O(nlogn)
 n = length of S
 l = 1
 r = n                                                  O(n)
 while l < r:
        t = S[l] + S[r]
        if t = k:
             return found and Yes                       ∴ O(nlogn)
        elif t < k:
             l += 1
        else:
             r -= 1
        return "No."
```

Two-set pairsum

Given sets X and Y of integers, a target integer k, is there x in X and y in Y so that $x+y=k$?

-> sort X and Y

-> compute $t=x_1$ and y_n (x_1 is first x in X, y_n is largest in Y)

$t=k$, found

$t<k$, remove x_1, $t=x_2+y_n$

$t>k$, remove y_n, $t=x_1+y_{n-1}$

Pseudocode

```
TwoSetPairsum(X, Y, k):

    n = length of X
    l = 1
    r = n

    while l <= n and r >= 1:
        t = X[l] + Y[r]
        if t = k:
            return found
        elif t < k:
            l += 1
        else:
            r -= 1

    return "No"
```

Subset Sum

$k = 10$

$[4, 3, 6, 1, 7]$

Subset sum = set S of n integers and target k, does S have a subset that sums to k?

-> is a decision problem
-> NP complete

Brute force:

$S = \{1, 2, 3\}$
subsets:
$\{\}, \{1\}, \{2\}, \{3\}$
$\{1,2\}, \{1,3\}, \{2,3\}$
$\{1,2,3\}$

$\therefore 2^n$, n = no. of elems
in big $O(2^n)$ -> very slow

algo DNC

```
subsetsum(S, k):
    split S into two S1 and S2
    Compute sums of all subsets of S1, -> A1
    Compute sums of all subsets of S2, -> A2

    if k ∈ A1 or k ∈ A2:
            return Yes
    else:
        TwoSetPairSum(A1, A2, k)
```