Recursion and Divide-and-Conquer

CS2800: Design and Analysis of Algorithms

Yadu Vasudev
yadu@cse.iitm.ac.in
IIT Madras

Module plan

1. Asymptotic analysis
2. Integer Multiplication Redux
3. Order statistics
4. Fast Fourier Transform
5. Closest pair of points

Asymptotic analysis

Recall: Asymptotic analysis

- Often, we are only interested in the the rate of growth of a function w.r.t to other functions

Recall: Asymptotic analysis

- Often, we are only interested in the the rate of growth of a function w.r.t to other functions
    - Does 2^n grow faster than n^{100}?
    - Does 2^n grow faster than $2^{0.9n}$?

Recall: Asymptotic analysis

- Often, we are only interested in the the rate of growth of a function w.r.t to other functions
  - Does 2^n grow faster than n^{100}?
  - Does 2^n grow faster than $2^{0.9n}$?
- For an algorithm $\mathcal{A}$, which function f(n) captures the rate of growth of the running time of $\mathcal{A}$ the best?

Recall: Asymptotic analysis

- Often, we are only interested in the the rate of growth of a function w.r.t to other functions
  - Does 2^n grow faster than n^{100}?
  - Does 2^n grow faster than $2^{0.9n}$?
- For an algorithm $\mathcal{A}$, which function $f(n)$ captures the rate of growth of the running time of $\mathcal{A}$ the best?

Definitions of O, Ω, and Θ

- A function $f(n) = O(g(n))$ if $\exists c > 0$ and $n_0 \geq 1$ such that $f(n) \leq cg(n), \forall n \geq n_0$
- A function $f(n) = \Omega(g(n))$ if $\exists c > 0$ and $n_0 \geq 1$ such that $f(n) \geq cg(n), \forall n \geq n_0$
- A function $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Recall: Asymptotic analysis

- Often, we are only interested in the the rate of growth of a function w.r.t to other functions
  - Does 2^n grow faster than n^{100}?
  - Does 2^n grow faster than $2^{0.9n}$?
- For an algorithm $\mathcal{A}$, which function $f(n)$ captures the rate of growth of the running time of $\mathcal{A}$ the best?

Little-oh

- A function $f(n) = o(g(n))$ if $\forall c > 0$, there exists an n_0 s.t $f(n) < cg(n), \forall n \geq n_0$

- Often, we are only interested in the the rate of growth of a function w.r.t to other functions
  - Does 2^n grow faster than n^{100}?
  - Does 2^n grow faster than $2^{0.9n}$?
- For an algorithm $\mathcal{A}$, which function $f(n)$ captures the rate of growth of the running time of $\mathcal{A}$ the best?

Little-oh

- A function $f(n) = o(g(n))$ if $\forall c > 0$, there exists an n_0 s.t $f(n) < cg(n), \forall n \geq n_0$
- Equivalently,

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Recall: Asymptotic analysis

- Often, we are only interested in the the rate of growth of a function w.r.t to other functions
  - Does 2^n grow faster than n^{100}?
  - Does 2^n grow faster than $2^{0.9n}$?
- For an algorithm $\mathcal{A}$, which function f(n) captures the rate of growth of the running time of $\mathcal{A}$ the best?

Little-oh

- A function $f(n) = o(g(n))$ if $\forall c > 0$, there exists an n_0 s.t $f(n) < cg(n), \forall n \geq n_0$
- Equivalently,

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

- f(n) is a strictly slower growing function than g(n)

Recall: Asymptotic analysis

- Often, we are only interested in the the rate of growth of a function w.r.t to other functions
    - Does 2^n grow faster than n^{100}?
    - Does 2^n grow faster than $2^{0.9n}$?
- For an algorithm $\mathcal{A}$, which function f(n) captures the rate of growth of the running time of $\mathcal{A}$ the best?

Little-omega

- A function $f(n) = \omega(g(n))$ if $\forall c > 0$, there exists an n_0 s.t $f(n) > cg(n), \forall n \geq n_0$
- Equivalently,
$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$
- f(n) is a strictly faster growing function than g(n)

Recall: Asymptotic analysis

- O is like $\leq$, o is like $<$, Θ is like $=$, but...

Recall: Asymptotic analysis

- O is like $\leq$, o is like $<$, Θ is like $=$, but...

Consider a hypothetical algorithm for checking primality of a number n

- If n is even (check the LSB), answer ”no”
- Else run an algorithm with running time $\log n$

Recall: Asymptotic analysis

- O is like $\leq$, o is like $<$, Θ is like $=$, but...

Consider a hypothetical algorithm for checking primality of a number n

- If n is even (check the LSB), answer "no"
- Else run an algorithm with running time $\log n$

$$f(n) = \begin{cases} 1 & \text{if n is even,} \\ \log n & \text{otherwise} \end{cases}$$

Recall: Asymptotic analysis

- O is like $\leq$, o is like $<$, Θ is like $=$, but...

Consider a hypothetical algorithm for checking primality of a number n

- If n is even (check the LSB), answer ”no”
- Else run an algorithm with running time $\log n$

$$f(n) = \begin{cases} 1 & \text{if } n \text{ is even,} \\ \log n & \text{otherwise} \end{cases}$$

- $f(n) = O(\log n)$, but not $\Theta(\log n)$
- $f(n) = \Omega(1)$, but not $\Omega(g(n))$ for any function that grows with n
- $f(n) \neq o(\log n)$

Some interesting functions

Some interesting functions

- non-elementary functions - $f(n) = 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}$: height of the tower of 2s depends on the input size

Some interesting functions

- non-elementary functions - $f(n) = 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}$: height of the tower of 2s depends on the input size
- iterated logarithm - $\log^* n$: number of times log has to be taken before n is at most 1
    - $\log^* n \leq 4$ for all practical values of n

Some interesting functions

- non-elementary functions - $f(n) = 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}$: height of the tower of 2s depends on the input size
- iterated logarithm - $\log^* n$: number of times log has to be taken before n is at most 1
    - $\log^* n \leq 4$ for all practical values of n
- Ackermann function
    - $\alpha(n, 0)$ - increment by 1
    - $\alpha(n, 1)$ - perform incremement n times: addition
    - $\alpha(n, 2)$ - perform addition n times: multiplication
    - $\alpha(n, 3)$ - perform multiplication n times: exponentiation
    - $\alpha(n, 4)$ - perform exponentiation n times: ? $\ldots$

Integer Multiplication Redux

Recall: Peasant multiplication

Recall: Peasant multiplication

Input: Integers a and b
Output: $c = a \cdot b$
Set $c \leftarrow 0$
repeat
  if a is odd **then** $c \leftarrow c + b$;
  $b \leftarrow 2 \cdot b$
  $a \leftarrow \lfloor \frac{a}{2} \rfloor$
until $a = 0$;

Running time:

Recall: Peasant multiplication

Input: Integers a and b
Output: $c = a \cdot b$
Set $c \leftarrow 0$
repeat
  if a is odd **then** $c \leftarrow c + b$;
  $b \leftarrow 2 \cdot b$
  $a \leftarrow \lfloor \frac{a}{2} \rfloor$
until $a = 0$;

Running time:

- The main while loop runs for $\log a$ many iterations

Recall: Peasant multiplication

Input: Integers a and b
Output: $c = a \cdot b$
Set $c \leftarrow 0$
repeat
  if a is odd **then** $c \leftarrow c + b$;
  $b \leftarrow 2 \cdot b$
  $a \leftarrow \lfloor \frac{a}{2} \rfloor$
until $a = 0$;

Running time:

- The main while loop runs for $\log a$ many iterations
- Each operation in the while loop takes at most $\log a + \log b$ steps

Recall: Peasant multiplication

Input: Integers a and b
Output: $c = a \cdot b$
Set $c \leftarrow 0$
repeat
  if a is odd **then** $c \leftarrow c + b$;
  $b \leftarrow 2 \cdot b$
  $a \leftarrow \lfloor \frac{a}{2} \rfloor$
until $a = 0$;

Running time:

- The main while loop runs for $\log a$ many iterations
- Each operation in the while loop takes at most $\log a + \log b$ steps
- $O(n^2)$ time to multiply two n-digit numbers

Faster integer multiplication

Recursive solution:

$$(10^{n/2}a + b)(10^{n/2}c + d) = 10^{n}(ac) + 10^{n/2}(ad + bc) + (bd)$$

Faster integer multiplication

Recursive solution:

$$(10^{n/2}a + b)(10^{n/2}c + d) = 10^n(ac) + 10^{n/2}(ad + bc) + (bd)$$

Numbers with n/2 digits

Faster integer multiplication

Recursive solution:

$$(10^{n/2}a + b)(10^{n/2}c + d) = 10^n(ac) + 10^{n/2}(ad + bc) + (bd)$$

Numbers with n/2 digits

Recursive calls

Faster integer multiplication

Recursive solution:

$$(10^{n/2}a + b)(10^{n/2}c + d) = 10^n(ac) + 10^{n/2}(ad + bc) + (bd)$$

Numbers with n/2 digits

Recursive calls

Correctness: From the definition of multiplication

Faster integer multiplication

Recursive solution:

$$(10^{n/2}a + b)(10^{n/2}c + d) = 10^{n}(ac) + 10^{n/2}(ad + bc) + (bd)$$

Numbers with n/2 digits

Recursive calls

Correctness: From the definition of multiplication

Running time?

Analyzing recurrence relations

$$(10^{n/2}a + b)(10^{n/2}c + d) = 10^{n}(ac) + 10^{n/2}(ad + bc) + (bd)$$

Analyzing recurrence relations

$$(10^{n/2}a + b)(10^{n/2}c + d) = 10^{n}(ac) + 10^{n/2}(ad + bc) + (bd)$$

$$T(n) = 4\ T(\frac{n}{2}) + O(n)$$

Analyzing recurrence relations

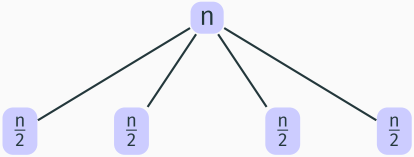
$$(10^{n/2}a + b)(10^{n/2}c + d) = 10^{n}(ac) + 10^{n/2}(ad + bc) + (bd)$$

$$\underbrace{T(n)}_{\text{running time on inputs of size n}} = \underbrace{4}_{\text{\# recursive calls}}\, T(\underbrace{\tfrac{n}{2}}_{\text{size of sub-problems}}) + \underbrace{O(n)}_{\text{adding the sub-products}}$$

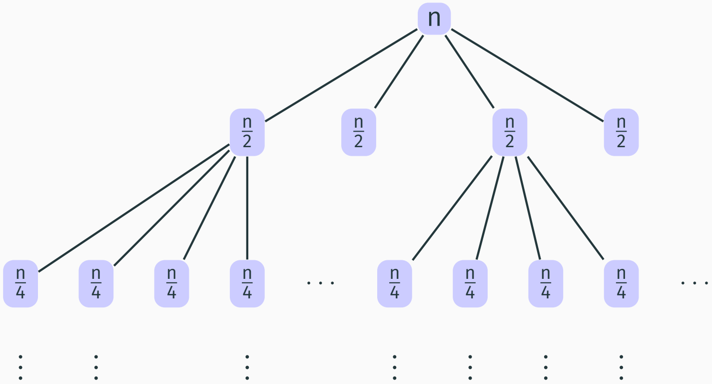
Analyzing recurrence relations: Recursion trees

n

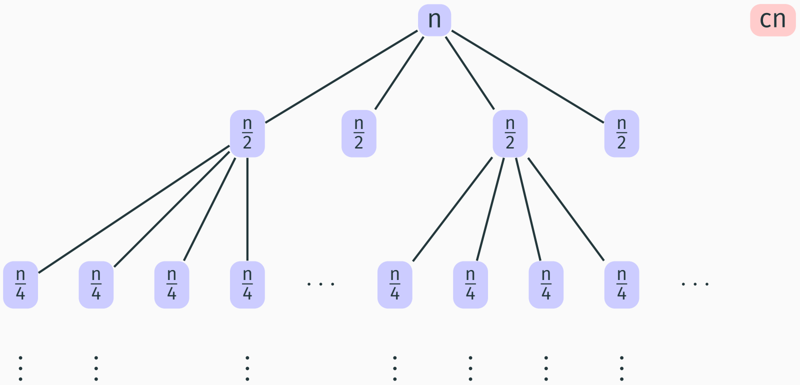
Analyzing recurrence relations: Recursion trees



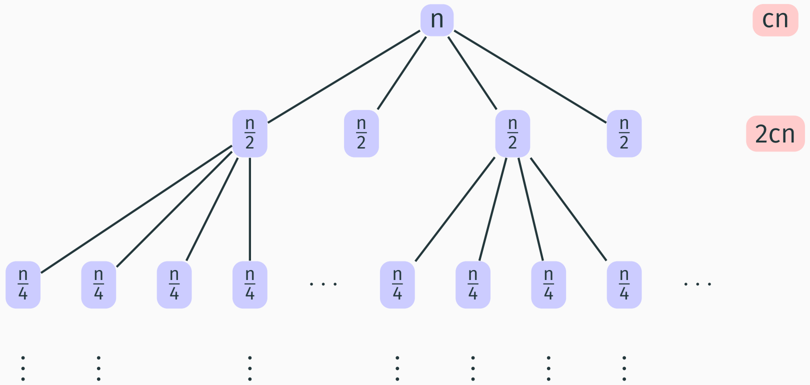
Analyzing recurrence relations: Recursion trees



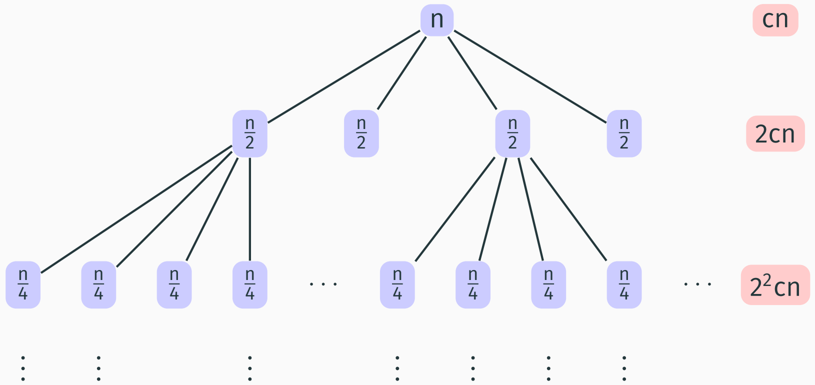
Analyzing recurrence relations: Recursion trees



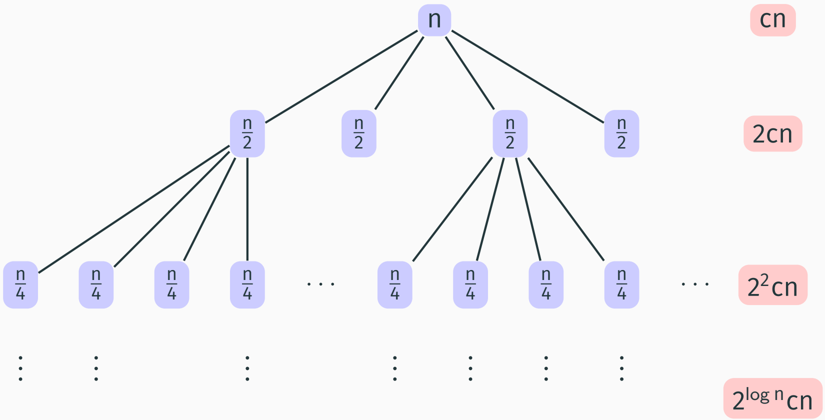
Analyzing recurrence relations: Recursion trees



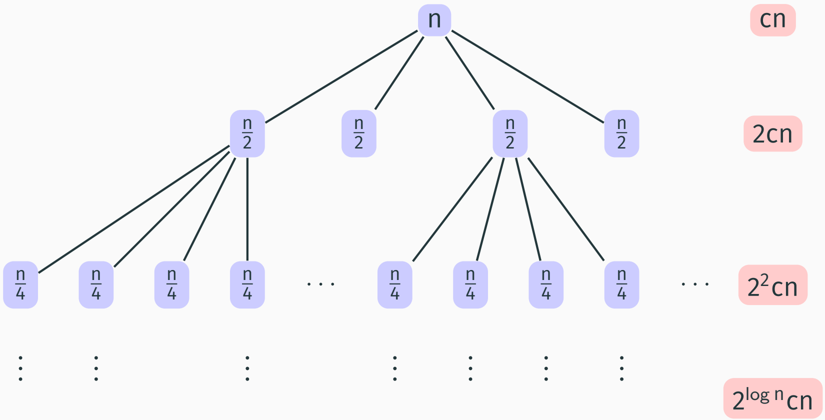
Analyzing recurrence relations: Recursion trees



Analyzing recurrence relations: Recursion trees



Analyzing recurrence relations: Recursion trees



Running time: $cn(1 + 2 + 2^2 + \cdots + 2^{\log n}) = O(n^2)$

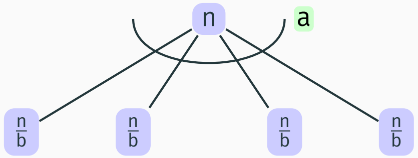
Recursion trees: The master theorem

Solving recurrence relations of the form $T(n) = aT(\frac{n}{b}) + f(n)$?

n

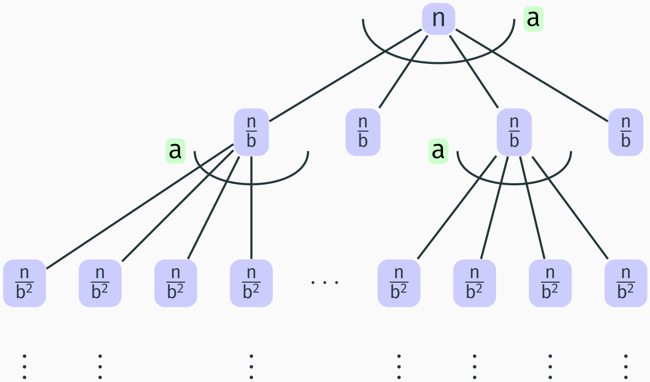
Recursion trees: The master theorem

Solving recurrence relations of the form $T(n) = aT(\frac{n}{b}) + f(n)$?

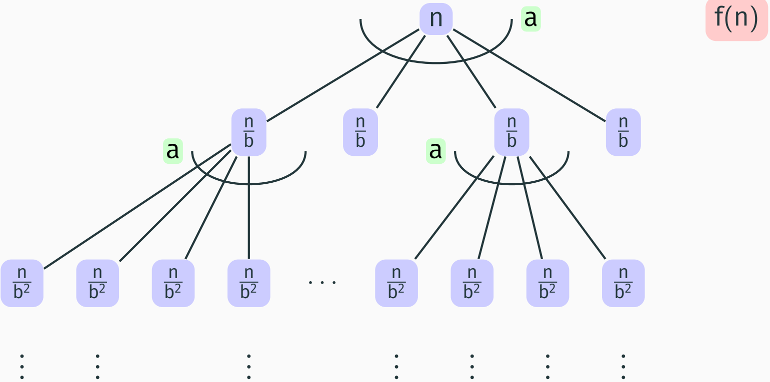


Recursion trees: The master theorem

Solving recurrence relations of the form $T(n) = aT(\frac{n}{b}) + f(n)$?

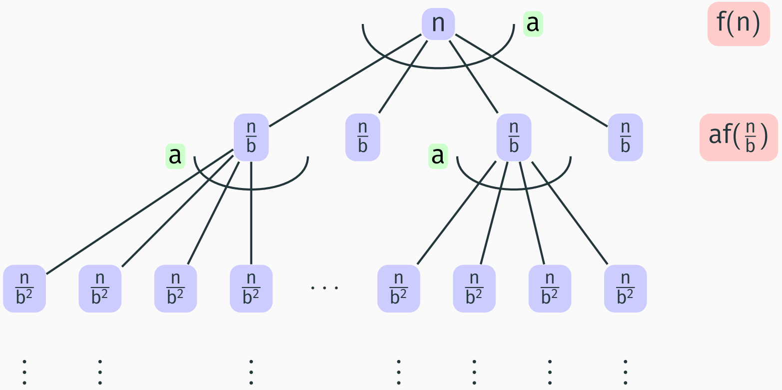


Solving recurrence relations of the form $T(n) = aT(\frac{n}{b}) + f(n)$?



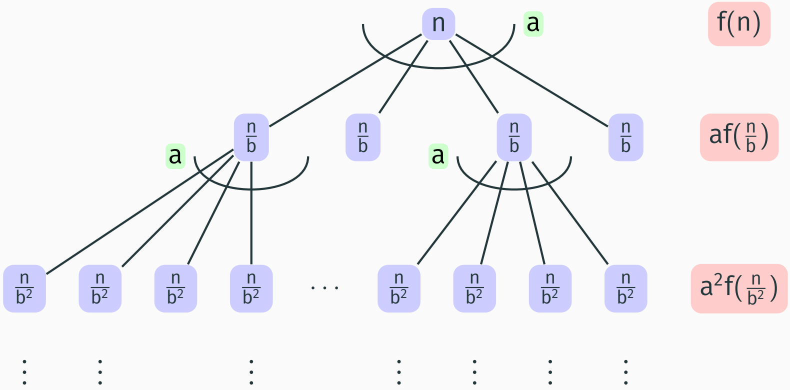
Recursion trees: The master theorem

Solving recurrence relations of the form $T(n) = aT(\frac{n}{b}) + f(n)$?



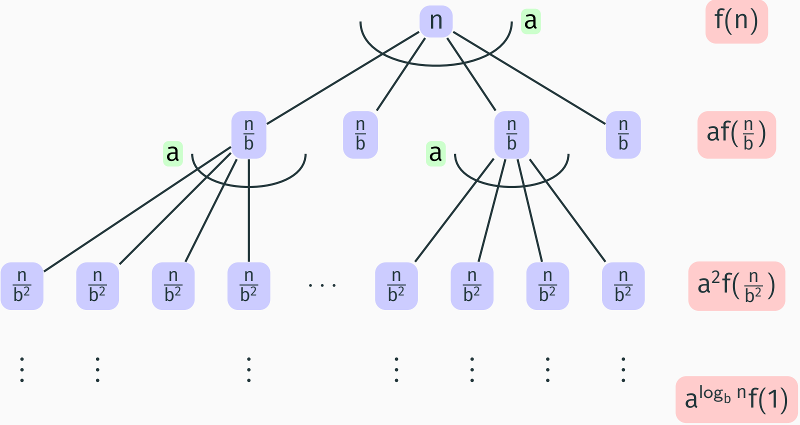
Recursion trees: The master theorem

Solving recurrence relations of the form $T(n) = aT(\frac{n}{b}) + f(n)$?



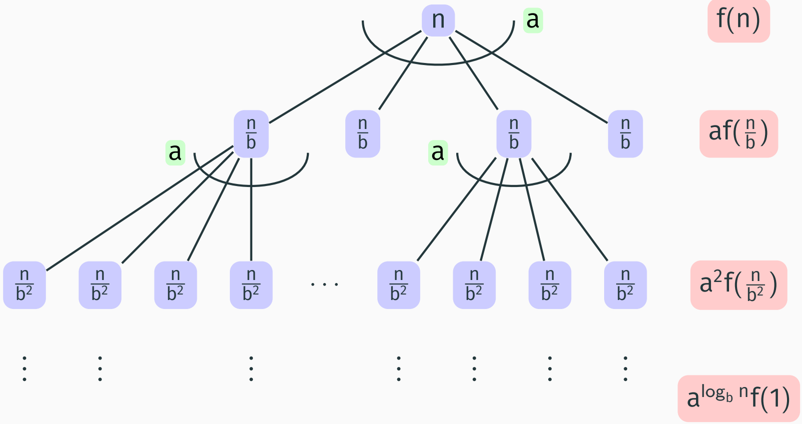
Recursion trees: The master theorem

Solving recurrence relations of the form $T(n) = aT(\frac{n}{b}) + f(n)$?



Recursion trees: The master theorem

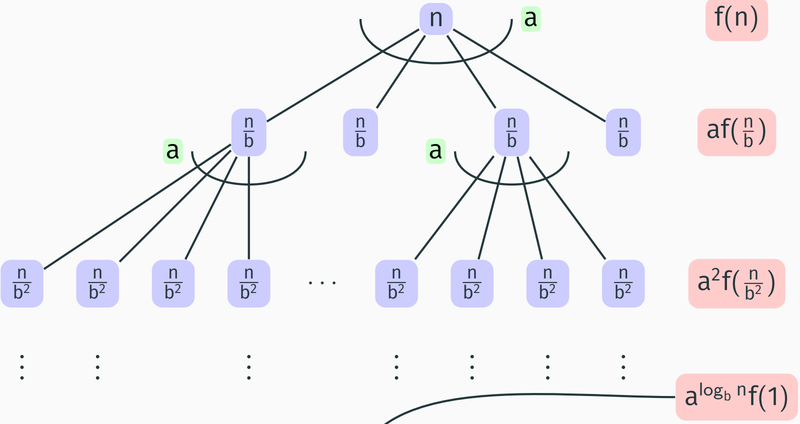
Solving recurrence relations of the form $T(n) = aT(\frac{n}{b}) + f(n)$?



$$T(n) = \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right)$$

Recursion trees: The master theorem

Solving recurrence relations of the form $T(n) = aT(\frac{n}{b}) + f(n)$?



$$T(n) = \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n-1} a^i f\left(\frac{n}{b^i}\right)$$

Recursion trees: The master theorem

When

$$T(n) = \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right),$$

Recursion trees: The master theorem

When

$$T(n) = \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right),$$

- If $f(n) = O(n^{\log_b a - \epsilon})$, then $T(n) = \Theta(n^{\log_b a})$
- If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
- If $f(n) = \Omega(n^{\log_b a + \epsilon})$ and $af(n/b) \leq cf(n)$ for some constant $c < 1$, then $T(n) = \Theta(f(n))$

Recursion trees: The master theorem

When

$$T(n) = \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right),$$

- If $f(n) = O(n^{\log_b a - \epsilon})$, then $T(n) = \Theta(n^{\log_b a})$
- If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
- If $f(n) = \Omega(n^{\log_b a + \epsilon})$ and $af(n/b) \leq cf(n)$ for some constant $c < 1$, then $T(n) = \Theta(f(n))$

Points to remember:

- Floors and ceilings are not important in proving the upper bounds (for most cases)
- Use recursion trees directly when the master theorem cannot be applied

Integer multiplication: Karatsuba's method

Integer multiplication: Karatsuba's method

Karatsuba's observation: $bc + ad = ac + bd - (a - b)(c - d)$

Integer multiplication: Karatsuba's method

Karatsuba's observation: $bc + ad = ac + bd - (a - b)(c - d)$

$$(10^{n/2}a + b)(10^{n/2}c + d) = 10^n(ac) + 10^{n/2}(ad + bc) + (bd)$$

Integer multiplication: Karatsuba's method

Karatsuba's observation: $bc + ad = ac + bd - (a-b)(c-d)$

$$(10^{n/2}a + b)(10^{n/2}c + d) = 10^{n}(ac) + 10^{n/2}(ad + bc) + (bd)$$

$$(a-b)(c-d)$$

Integer multiplication: Karatsuba's method

Karatsuba's observation: $bc + ad = ac + bd - (a - b)(c - d)$

$$(10^{n/2}a + b)(10^{n/2}c + d) = 10^n(ac) + 10^{n/2}(ad + bc) + (bd)$$

$$(a - b)(c - d)$$

Running time: $T(n) = 3T(n/2) + O(n)$

Integer multiplication: Karatsuba's method

Karatsuba's observation: $bc + ad = ac + bd - (a - b)(c - d)$

$$(10^{n/2}a + b)(10^{n/2}c + d) = 10^n(ac) + 10^{n/2}(ad + bc) + (bd)$$

$$(a - b)(c - d)$$

Running time: $T(n) = 3T(n/2) + O(n)$

Is the master theorem applicable here?

Integer multiplication: Karatsuba's method

Karatsuba's observation: $bc + ad = ac + bd - (a - b)(c - d)$

$$(10^{n/2}a + b)(10^{n/2}c + d) = 10^n(ac) + 10^{n/2}(ad + bc) + (bd)$$

$$(a - b)(c - d)$$

Running time: $T(n) = 3T(n/2) + O(n)$

Is the master theorem applicable here? $T(n) = \Theta(n^{\log 3})$

Integer multiplication: A short history

Integer multiplication: A short history

Peasant multiplication - $O(n^2)$	1600 BC
Lattice multiplication - $O(n^2)$	1200
Karatsuba's method - $O(n^{\log 3})$	1960
Schönhage-Strassen - $O(n \log n \log\log n)$	1971
Conjecture: $\Theta(n \log n)$	
Fürer/De-Saha-Kurur-Saptharishi - $O(2^{\Theta(\log^* n)} n \log n)$	2008
Harvey-van der Hoeven - $O(2^{2\log^* n} n \log n)$	2018
Harvey-van der Hoeven - $O(n \log n)$	2019

"...our work is expected to be the end of the road for this problem, although we don't know yet how to prove this rigorously."

Order statistics

Order statistics

Given an array $A[1, 2, \ldots, n]$, find the k^{th} element in the sorted array

Order statistics

Given an array $A[1, 2, \ldots, n]$, find the k^{th} element in the sorted array

- Sort the array, and return $A[k]$

Order statistics

Given an array $A[1, 2, \ldots, n]$, find the k^{th} element in the sorted array

- Sort the array, and return $A[k] \longrightarrow O(n \log n)$

Order statistics

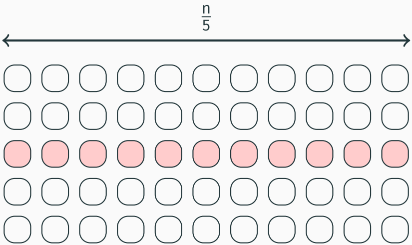
Given an array $A[1, 2, \ldots, n]$, find the k^{th} element in the sorted array

- Sort the array, and return $A[k] \longrightarrow O(n \log n)$
- If we find a pivot partitioning A into two parts of size αn and $(1 - \alpha)n$ in $O(n)$ time, then we have a recursive algorithm with running time

$$T(n) = T(\alpha n) + O(n)$$

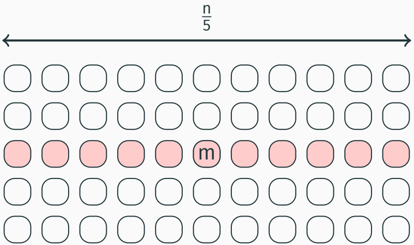
Median of medians

Median of medians



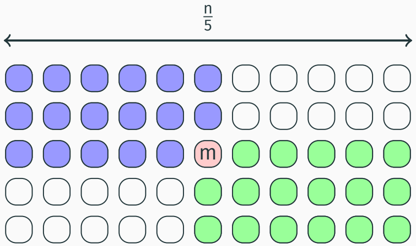
- Divide the array A into n/5 subarrays of length 5 and find their medians

Median of medians



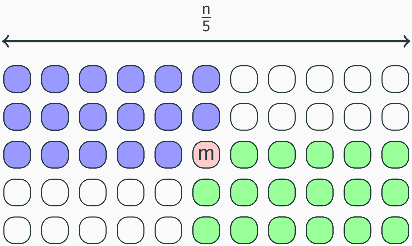
- Divide the array A into n/5 subarrays of length 5 and find their medians
- How good is the median of these medians as a pivot?

Median of medians



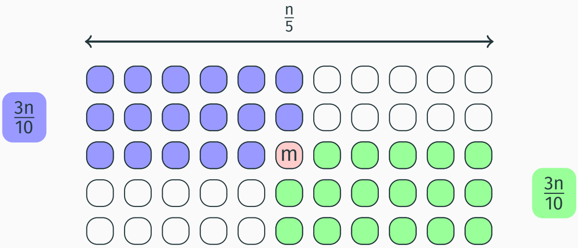
- Divide the array A into n/5 subarrays of length 5 and find their medians
- How good is the median of these medians as a pivot?

Median of medians



- Divide the array A into n/5 subarrays of length 5 and find their medians
- How good is the median of these medians as a pivot?
- All the blue elements are $\leq$ m and all the green elements are $\geq$ m - How many are there?

Median of medians



- Divide the array A into n/5 subarrays of length 5 and find their medians
- How good is the median of these medians as a pivot?
- All the blue elements are $\leq$ m and all the green elements are $\geq$ m - How many are there?

A recursive algorithm

Input: Array $A[1, 2, \ldots, n]$, k

if $n \leq 10$ **then**
    brute-force search
else
    set $d \leftarrow n/5$
    for $1 \leq i \leq d$ **do**
        set $M[i] \leftarrow$ median of $A[5i - 4, \ldots, 5i]$
    set $m \leftarrow$ median of M recursively
    $r \leftarrow \textsc{Partition}(A[1, 2, \ldots, n], m)$
    if $k < r$ **then**
        recursively search in $A[1, 2, \ldots, m - 1]$ for k
    else
        if $k > r$ **then**
            recursively search in $A[m + 1, \ldots, n]$ for $k - r$
        else
            return m

A recursive algorithm

Input: Array $A[1, 2, \ldots, n]$, k

if $n \leq 10$ **then**

  brute-force search

else

  set $d \leftarrow n/5$

  for $1 \leq i \leq d$ **do**

    set $M[i] \leftarrow$ median of $A[5i - 4, \ldots, 5i]$

  set $m \leftarrow$ median of M recursively $T(\frac{n}{5})$

  $r \leftarrow$ PARTITION$(A[1, 2, \ldots, n], m)$ $O(n)$

  if $k < r$ **then**

    recursively search in $A[1, 2, \ldots, m - 1]$ for k

  else $T(\frac{7n}{10})$

    if $k > r$ **then**

      recursively search in $A[m + 1, \ldots, n]$ for $k - r$

    else

      return m

Running time - median of medians algorithm

- The running time is given by

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$$

Running time - median of medians algorithm

- The running time is given by

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$$

- What happens if we write this as

$$T(n) \leq 2T\left(\frac{7n}{10}\right) + O(n)?$$

Running time - median of medians algorithm

- The running time is given by

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$$

- What happens if we write this as

$$T(n) \leq 2T\left(\frac{7n}{10}\right) + O(n)?$$

$O(n^{\log_{10/7} 2})$

Running time - median of medians algorithm

- The running time is given by

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$$

- What happens if we write this as

$$T(n) \leq 2T\left(\frac{7n}{10}\right) + O(n)?$$

$O(n^{\log_{10/7} 2})$

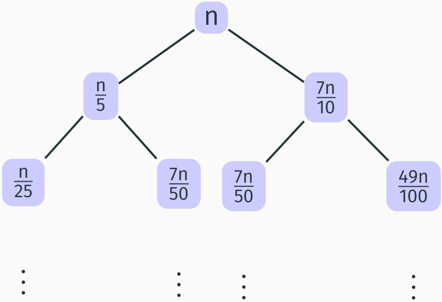
- Is our algorithm bad or analysis not tight?

Running time using recursion trees

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$$

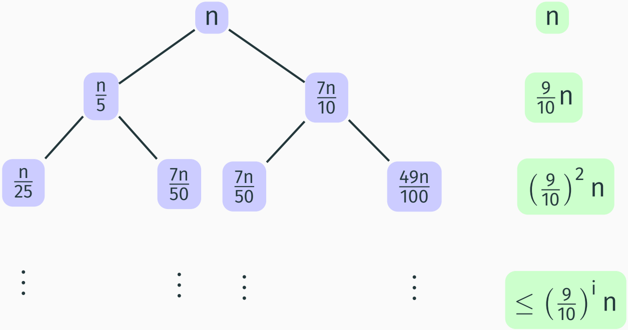
Running time using recursion trees

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$$



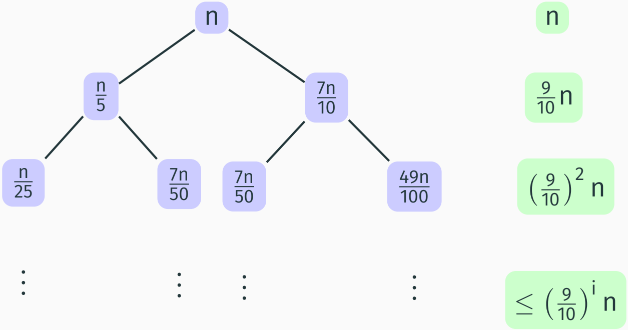
Running time using recursion trees

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$$



Running time using recursion trees

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n)$$



$$T(n) \leq \left(1 + \frac{9}{10} + \left(\frac{9}{10}\right)^2 + \cdots\right) n = O(n)$$

Fast Fourier Transform

Convolution of vectors

Given $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$, $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1})$, compute the value

$$\mathbf{a} * \mathbf{b} = (c_0, c_1, \ldots, c_{2n-1}), \text{ where}$$
$$c_k = \sum_{i+j=k} a_i b_j$$

Convolution of vectors

Given $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$, $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1})$, compute the value

$$\mathbf{a} * \mathbf{b} = (c_0, c_1, \ldots, c_{2n-1}), \text{ where}$$
$$c_k = \sum_{i+j=k} a_i b_j$$

- **Polynomial multiplication:** If $\mathbf{a}$ and $\mathbf{b}$ represent the coefficients of the polynomials, then the convolution is the coefficients of their product

Convolution of vectors

Given $\mathbf{a} = (a_0, a_1, \dots, a_{n-1})$, $\mathbf{b} = (b_0, b_1, \dots, b_{n-1})$, compute the value

$$\mathbf{a} * \mathbf{b} = (c_0, c_1, \dots, c_{2n-1}), \text{ where}$$
$$c_k = \sum_{i+j=k} a_i b_j$$

- **Polynomial multiplication:** If $\mathbf{a}$ and $\mathbf{b}$ represent the coefficients of the polynomials, then the convolution is the coefficients of their product

$$\mathbf{a} = (2, 3, 4) \equiv 2 + 3x + 4x^2$$
$$\mathbf{b} = (1, 2, 3) \equiv 1 + 2x + 3x^2$$
$$\mathbf{a} * \mathbf{b} = (2, 7, 16, 17, 12) \equiv 2 + 7x + 16x^2 + 17x^3 + 12x^4$$

Convolution of vectors

Given $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$, $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1})$, compute the value

$$\mathbf{a} * \mathbf{b} = (c_0, c_1, \ldots, c_{2n-1}), \text{ where}$$
$$c_k = \sum_{i+j=k} a_i b_j$$

- **Polynomial multiplication:** If $\mathbf{a}$ and $\mathbf{b}$ represent the coefficients of the polynomials, then the convolution is the coefficients of their product

$$\mathbf{a} = (2, 3, 4) \equiv 2 + 3x + 4x^2$$
$$\mathbf{b} = (1, 2, 3) \equiv 1 + 2x + 3x^2$$
$$\mathbf{a} * \mathbf{b} = (2, 7, 16, 17, 12) \equiv 2 + 7x + 16x^2 + 17x^3 + 12x^4$$

- **Signal processing:** - smoothing discrete measurements, analyzing response of a linear time-invariant system

Convolution of vectors

The naive algorithm:

Convolution of vectors

The naive algorithm:

- Compute c_k using at most k multiplications and additions, for $k \in \{0, 1, \ldots, 2n-2\}$

$$c_k = \sum_{i+j=k} a_i b_j$$

Convolution of vectors

The naive algorithm:

- Compute c_k using at most k multiplications and additions, for $k \in \{0, 1, \ldots, 2n-2\}$

$$c_k = \sum_{i+j=k} a_i b_j$$

- **Running time:** $O(n^2)$

Convolution of vectors

The naive algorithm:

- Compute c_k using at most k multiplications and additions, for $k \in \{0, 1, \ldots, 2n-2\}$

$$c_k = \sum_{i+j=k} a_i b_j$$

- **Running time:** $O(n^2)$

Fast Fourier transform:

Convolution of vectors

The naive algorithm:

- Compute c_k using at most k multiplications and additions, for $k \in \{0, 1, \ldots, 2n-2\}$

$$c_k = \sum_{i+j=k} a_i b_j$$

- **Running time:** $O(n^2)$

Fast Fourier transform:

- Running time of $O(n \log n)$

Convolution of vectors

The naive algorithm:

- Compute c_k using at most k multiplications and additions, for $k \in \{0, 1, \ldots, 2n-2\}$

$$c_k = \sum_{i+j=k} a_i b_j$$

- **Running time:** $O(n^2)$

Fast Fourier transform:

- Running time of $O(n \log n)$
- First described by Gauss in 1805
- Modern version by Cooley and Tukey in 1965

Vectors and polynomials

Convolution $\equiv$ Polynomial multiplication

Convolution $\equiv$ Polynomial multiplication

$$P(x) = \sum_{i=0}^{n-1} a_i x^i,\ Q(x) = \sum_{i=0}^{n-1} b_i x^i \Rightarrow \mathbf{a} * \mathbf{b} \equiv P(x)Q(x)$$

Convolution $\equiv$ Polynomial multiplication

$P(x) = \sum_{i=0}^{n-1} a_i x^i$, $Q(x) = \sum_{i=0}^{n-1} b_i x^i \Rightarrow \mathbf{a} * \mathbf{b} \equiv P(x)Q(x)$

Naive algorithm: Multiply $a_i x^i$ with $Q(x)$ for each i and then add

Polynomials - an alternate view

A polynomial $P(x)$ of degree $n - 1$ is uniquely defined by its evaluation on n points

Polynomials - an alternate view

A polynomial $P(x)$ of degree $n-1$ is uniquely defined by its evaluation on n points

$$P(x) = \sum_{i=0}^{n-1} a_i x^i$$

$$\text{vector of coefficients} \longleftarrow \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} \equiv \begin{pmatrix} P(\alpha_0) \\ P(\alpha_1) \\ P(\alpha_2) \\ \vdots \\ P(\alpha_{n-1}) \end{pmatrix} \longrightarrow \text{vector of evaluations}$$

Polynomials - an alternate view

A polynomial $P(x)$ of degree $n - 1$ is uniquely defined by its evaluation on n points

$$P(x) = \sum_{i=0}^{n-1} a_i x^i \quad Q(x) = \sum_{i=0}^{n-1} b_i x^i$$

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} * \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{pmatrix} \equiv \begin{pmatrix} P(\alpha_0)Q(\alpha_0) \\ P(\alpha_1)Q(\alpha_1) \\ P(\alpha_2)Q(\alpha_2) \\ \vdots \\ P(\alpha_{2n-2})Q(\alpha_{2n-2}) \end{pmatrix}$$

Convolution - outline of an algorithm

$$P(x) = \sum_{i=0}^{n-1} a_i x^i \quad Q(x) = \sum_{i=0}^{n-1} b_i x^i$$

Convolution - outline of an algorithm

$$P(x) = \sum_{i=0}^{n-1} a_i x^i \quad Q(x) = \sum_{i=0}^{n-1} b_i x^i$$

$$(a_0, a_1, \ldots, a_{n-1})$$
$$(b_0, b_1, \ldots, b_{n-1})$$

Convolution - outline of an algorithm

$$P(x) = \sum_{i=0}^{n-1} a_i x^i \quad Q(x) = \sum_{i=0}^{n-1} b_i x^i$$

$$\begin{matrix} (a_0, a_1, \ldots, a_{n-1}) \\ (b_0, b_1, \ldots, b_{n-1}) \end{matrix} \xrightarrow{\text{evaluation}} \begin{matrix} P(\alpha_0), P(\alpha_1), \ldots, P(\alpha_{2n-2}) \\ Q(\alpha_0), Q(\alpha_1), \ldots, Q(\alpha_{2n-2}) \end{matrix}$$

Convolution - outline of an algorithm

$$P(x) = \sum_{i=0}^{n-1} a_i x^i \quad Q(x) = \sum_{i=0}^{n-1} b_i x^i$$

$$\begin{array}{c} (a_0, a_1, \ldots, a_{n-1}) \\ (b_0, b_1, \ldots, b_{n-1}) \end{array} \xrightarrow{\text{evaluation}} \begin{array}{c} P(\alpha_0), P(\alpha_1), \ldots, P(\alpha_{2n-2}) \\ Q(\alpha_0), Q(\alpha_1), \ldots, Q(\alpha_{2n-2}) \end{array}$$

$$\Big\downarrow \text{pointwise product}$$

$$\begin{array}{c} P(\alpha_0)Q(\alpha_0), \\ P(\alpha_1)Q(\alpha_1), \\ \ldots \\ P(\alpha_{2n-2})Q(\alpha_{2n-2}) \end{array}$$

Convolution - outline of an algorithm

$$P(x) = \sum_{i=0}^{n-1} a_i x^i \quad Q(x) = \sum_{i=0}^{n-1} b_i x^i$$

$$\begin{tikzcd}
\begin{matrix}(a_0, a_1, \ldots, a_{n-1}) \\ (b_0, b_1, \ldots, b_{n-1})\end{matrix} \arrow[r, "\text{evaluation}"] & \begin{matrix}P(\alpha_0), P(\alpha_1), \ldots, P(\alpha_{2n-2}) \\ Q(\alpha_0), Q(\alpha_1), \ldots, Q(\alpha_{2n-2})\end{matrix} \arrow[d, "\text{pointwise product}"'] \\
P(x)Q(x) & \begin{matrix}P(\alpha_0)Q(\alpha_0), \\ P(\alpha_1)Q(\alpha_1), \\ \ldots \\ P(\alpha_{2n-2})Q(\alpha_{2n-2})\end{matrix} \arrow[l, "\text{interpolation}"']
\end{tikzcd}$$

Polynomial evaluation

Polynomial evaluation

Naive evaluation of a polynomial on $O(n)$ points requires $O(n^2)$ time

Polynomial evaluation

Naive evaluation of a polynomial on $O(n)$ points requires $O(n^2)$ time

Can we do better by choosing α_is carefully?

Polynomial evaluation

Naive evaluation of a polynomial on $O(n)$ points requires $O(n^2)$ time

Can we do better by choosing α_is carefully?

$$P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots$$

Polynomial evaluation

Naive evaluation of a polynomial on $O(n)$ points requires $O(n^2)$ time

Can we do better by choosing α_is carefully?

$$\begin{aligned} P(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots \\ &= (a_0 + a_2 x^2 + a_4 x^4 + \cdots) + x(a_1 + a_3 x^2 + a_5 x^4 + \cdots) \end{aligned}$$

Polynomial evaluation

Naive evaluation of a polynomial on $O(n)$ points requires $O(n^2)$ time

Can we do better by choosing α_is carefully?

$$\begin{aligned} P(x) &= a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots \\ &= (a_0 + a_2x^2 + a_4x^4 + \cdots) + x(a_1 + a_3x^2 + a_5x^4 + \cdots) \end{aligned}$$

$P_e(x^2)$ - polynomial of degree at most $\frac{n}{2}$

$P_o(x^2)$ - polynomial of degree at most $\frac{n}{2}$

Polynomial evaluation

$$P(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots$$
$$= (a_0 + a_2x^2 + a_4x^4 + \cdots) + x(a_1 + a_3x^2 + a_5x^4 + \cdots)$$

$P_e(x^2)$ - polynomial of degree at most $\frac{n}{2}$

$P_o(x^2)$ - polynomial of degree at most $\frac{n}{2}$

Polynomial evaluation

$$P(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots$$
$$= (a_0 + a_2 x^2 + a_4 x^4 + \cdots) + x(a_1 + a_3 x^2 + a_5 x^4 + \cdots)$$

$P_e(x^2)$ - polynomial of degree at most $\frac{n}{2}$

$P_o(x^2)$ - polynomial of degree at most $\frac{n}{2}$

$$\pm\alpha_0, \pm\alpha_1, \pm\alpha_2, \ldots, \pm\alpha_{n/2-1}$$

Polynomial evaluation

$$P(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots$$
$$= (a_0 + a_2x^2 + a_4x^4 + \cdots) + x(a_1 + a_3x^2 + a_5x^4 + \cdots)$$

$P_e(x^2)$ - polynomial of degree at most $\frac{n}{2}$

$P_o(x^2)$ - polynomial of degree at most $\frac{n}{2}$

$$\pm\alpha_0, \pm\alpha_1, \pm\alpha_2, \ldots, \pm\alpha_{n/2-1}$$

$$P(\alpha_i) = P_e(\alpha_i^2) + \alpha_i P_o(\alpha_i^2)$$
$$P(-\alpha_i) = P_e(\alpha_i^2) - \alpha_i P_o(\alpha_i^2)$$

Polynomial evaluation

$$
\begin{aligned}
P(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots \\
&= (a_0 + a_2 x^2 + a_4 x^4 + \cdots) + x(a_1 + a_3 x^2 + a_5 x^4 + \cdots)
\end{aligned}
$$

$P_e(x^2)$ - polynomial of degree at most $\frac{n}{2}$

$P_o(x^2)$ - polynomial of degree at most $\frac{n}{2}$

$$\pm\alpha_0, \pm\alpha_1, \pm\alpha_2, \ldots, \pm\alpha_{n/2-1}$$

$$
\begin{aligned}
P(\alpha_i) &= P_e(\alpha_i^2) + \alpha_i P_o(\alpha_i^2) \\
P(-\alpha_i) &= P_e(\alpha_i^2) - \alpha_i P_o(\alpha_i^2)
\end{aligned}
$$

evaluation of two $n/2$-degree polynomials at $n/2$ points

Polynomial evaluation - a “complex” approach

$$P(\alpha_i) = P_e(\alpha_i^2) + \alpha_i P_o(\alpha_i^2)$$
$$P(-\alpha_i) = \underbrace{P_e(\alpha_i^2)}_{\downarrow} - \underbrace{\alpha_i P_o(\alpha_i^2)}_{\downarrow}$$

evaluation of two n/2-degree polynomials at n/2 points

A divide-and-conquer algorithm works if α_i^2 are plus-minus pairs!

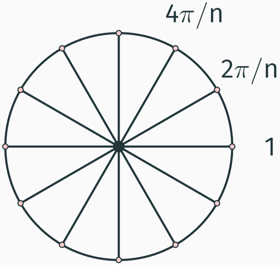
Polynomial evaluation - a “complex” approach

$$P(\alpha_i) = \boxed{P_e(\alpha_i^2)} + \boxed{\alpha_i P_o(\alpha_i^2)}$$
$$P(-\alpha_i) = \boxed{P_e(\alpha_i^2)} - \boxed{\alpha_i P_o(\alpha_i^2)}$$

evaluation of two n/2-degree polynomials at n/2 points

A divide-and-conquer algorithm works if α_i^2 are plus-minus pairs!

Complex roots of unity:



Polynomial evaluation - a “complex” approach

$$P(\alpha_i) = \boxed{P_e(\alpha_i^2)} + \boxed{\alpha_i P_o(\alpha_i^2)}$$
$$P(-\alpha_i) = \boxed{P_e(\alpha_i^2)} - \boxed{\alpha_i P_o(\alpha_i^2)}$$

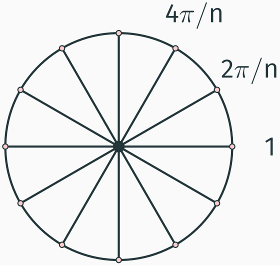
evaluation of two n/2-degree polynomials at n/2 points

A divide-and-conquer algorithm works if α_i^2 are plus-minus pairs!

Complex roots of unity:

$1, \omega, \omega^2, \ldots, \omega^i, \ldots, \omega^{n-1}$

$e^{2\pi i/n}$

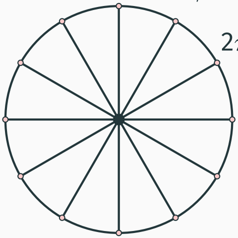


Polynomial evaluation - a "complex" approach

Complex roots of unity:

$$1, \omega, \omega^2, \ldots, \omega^j, \ldots, \omega^{n-1}$$

$$\omega \to e^{2\pi i/n}$$

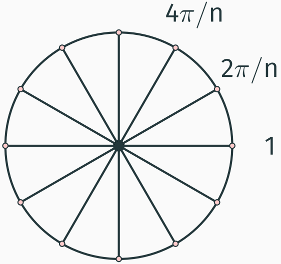


Polynomial evaluation - a “complex” approach

Complex roots of unity:

$1, \omega, \omega^2, \ldots, \omega^j, \ldots, \omega^{n-1}$

$e^{2\pi i/n}$



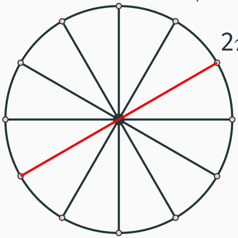
- For each j, $\omega^{\frac{n}{2}+j} = -\omega^j$ when n is even

Polynomial evaluation - a “complex” approach

Complex roots of unity:

$$1, \omega, \omega^2, \ldots, \omega^j, \ldots, \omega^{n-1}$$

$$\omega \rightarrow e^{2\pi i/n}$$



- For each j, $\omega^{\frac{n}{2}+j} = -\omega^j$ when n is even

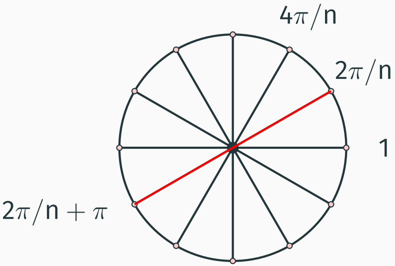
$$\omega^{\frac{n}{2}+j} = e^{\frac{2\pi i}{n}(\frac{n}{2}+j)=e^{\pi i}(e^{\frac{2\pi i}{n}})^j} = -\omega^j$$

Polynomial evaluation - a “complex” approach

Complex roots of unity:

$1, \omega, \omega^2, \ldots, \omega^j, \ldots, \omega^{n-1}$

$e^{2\pi i/n}$



- For each j, $\omega^{\frac{n}{2}+j} = -\omega^j$ when n is even

$$\omega^{\frac{n}{2}+j} = e^{\frac{2\pi i}{n}(\frac{n}{2}+j)=e^{\pi i}(e^{\frac{2\pi i}{n}})^j} = -\omega^j$$

- $(\omega^j)^2$ are $\frac{n}{2}$-th roots of unity

Polynomial evaluation - the Fast Fourier Transform

Polynomial evaluation - the Fast Fourier Transform

Input: polynomial $P(x)$ as a coefficient vector $(a_0, a_1, \ldots, a_{n-1})$
$(n = 2^k)$, ω - n^{th}root of unity
Output: $P(\omega^0), P(\omega), P(\omega^2), \ldots, P(\omega^{n-1})$

Polynomial evaluation - the Fast Fourier Transform

Input: polynomial $P(x)$ as a coefficient vector $(a_0, a_1, \ldots, a_{n-1})$
$(n = 2^k)$, ω - n^{th}root of unity
Output: $P(\omega^0), P(\omega), P(\omega^2), \ldots, P(\omega^{n-1})$
if $\omega = 1$ **then return** $P(1) \longrightarrow$ base case of recursion

Polynomial evaluation - the Fast Fourier Transform

Input: polynomial $P(x)$ as a coefficient vector $(a_0, a_1, \ldots, a_{n-1})$
$(n = 2^k)$, ω - n^{th}root of unity

Output: $P(\omega^0), P(\omega), P(\omega^2), \ldots, P(\omega^{n-1})$

if $\omega = 1$ **then return** $P(1) \longrightarrow$ base case of recursion

express $P(x) = P_e(x^2) + xP_o(x^2)$

Polynomial evaluation - the Fast Fourier Transform

Input: polynomial $P(x)$ as a coefficient vector $(a_0, a_1, \ldots, a_{n-1})$
$(n = 2^k)$, ω - n^{th}root of unity
Output: $P(\omega^0), P(\omega), P(\omega^2), \ldots, P(\omega^{n-1})$
if $\omega = 1$ **then return** $P(1) \longrightarrow$ base case of recursion
express $P(x) = P_e(x^2) + xP_o(x^2)$
Evaluate $P_e(\omega^2)$
Evaluate $P_o(\omega^2) \longrightarrow$ recursive calls - divide step

Polynomial evaluation - the Fast Fourier Transform

Input: polynomial $P(x)$ as a coefficient vector $(a_0, a_1, \ldots, a_{n-1})$
$(n = 2^k)$, ω - n^{th}root of unity
Output: $P(\omega^0), P(\omega), P(\omega^2), \ldots, P(\omega^{n-1})$
if $\omega = 1$ **then return** $P(1)$ $\longrightarrow$ base case of recursion
express $P(x) = P_e(x^2) + xP_o(x^2)$
Evaluate $P_e(\omega^2)$
Evaluate $P_o(\omega^2)$ $\longrightarrow$ recursive calls - divide step
foreach $j \in \{0, 1, \ldots, n-1\}$ **do**
$\quad P(\omega^j) = P_e(\omega^{2j}) + \omega^j P_o(\omega^{2j})$

combining solutions - conquer step

Polynomial evaluation - the Fast Fourier Transform

Input: polynomial $P(x)$ as a coefficient vector $(a_0, a_1, \ldots, a_{n-1})$
$(n = 2^k)$, ω - n^{th} root of unity
Output: $P(\omega^0), P(\omega), P(\omega^2), \ldots, P(\omega^{n-1})$
if $\omega = 1$ **then return** $P(1)$ ⟶ base case of recursion $O(n)$
express $P(x) = P_e(x^2) + xP_o(x^2)$
Evaluate $P_e(\omega^2)$
Evaluate $P_o(\omega^2)$ ⟶ recursive calls - divide step
foreach $j \in \{0, 1, \ldots, n-1\}$ **do**
    $P(\omega^j) = P_e(\omega^{2j}) + \omega^j P_o(\omega^{2j})$

combining solutions - conquer step

Polynomial evaluation - the Fast Fourier Transform

Input: polynomial $P(x)$ as a coefficient vector $(a_0, a_1, \ldots, a_{n-1})$
$(n = 2^k)$, ω - n^{th}root of unity

Output: $P(\omega^0), P(\omega), P(\omega^2), \ldots, P(\omega^{n-1})$

if $\omega = 1$ **then return** $P(1)$ $\longrightarrow$ base case of recursion $O(n)$

express $P(x) = P_e(x^2) + xP_o(x^2)$

Evaluate $P_e(\omega^2)$

Evaluate $P_o(\omega^2)$ $\longrightarrow$ recursive calls - divide step $2T(\frac{n}{2})$

foreach $j \in \{0, 1, \ldots, n-1\}$ **do**

$\quad P(\omega^j) = P_e(\omega^{2j}) + \omega^j P_o(\omega^{2j})$

combining solutions - conquer step

Polynomial evaluation - the Fast Fourier Transform

Input: polynomial $P(x)$ as a coefficient vector $(a_0, a_1, \ldots, a_{n-1})$
$(n = 2^k)$, ω - n^{th}root of unity

Output: $P(\omega^0), P(\omega), P(\omega^2), \ldots, P(\omega^{n-1})$

if $\omega = 1$ **then return** $P(1) \longrightarrow$ base case of recursion $O(n)$

express $P(x) = P_e(x^2) + xP_o(x^2)$

Evaluate $P_e(\omega^2)$

Evaluate $P_o(\omega^2) \longrightarrow$ recursive calls - divide step $2T(\frac{n}{2})$

foreach $j \in \{0, 1, \ldots, n-1\}$ **do**

$P(\omega^j) = P_e(\omega^{2j}) + \omega^j P_o(\omega^{2j})$

$O(n)$

combining solutions - conquer step

Where are we now?

$$P(x) = \sum_{i=0}^{n-1} a_i x^i \quad Q(x) = \sum_{i=0}^{n-1} b_i x^i$$

$$\begin{tikzcd}
\begin{matrix}(a_0, a_1, \ldots, a_{n-1})\\(b_0, b_1, \ldots, b_{n-1})\end{matrix} \arrow[r, "\text{evaluation}", "O(n \log n)"'] & \begin{matrix}P(\alpha_0), P(\alpha_1), \ldots, P(\alpha_{2n-2})\\Q(\alpha_0), Q(\alpha_1), \ldots, Q(\alpha_{2n-2})\end{matrix} \arrow[d, "\text{pointwise product}"', "O(n)"] \\
P(x)Q(x) & \begin{matrix}P(\alpha_0)Q(\alpha_0),\\P(\alpha_1)Q(\alpha_1),\\\ldots\\P(\alpha_{2n-2})Q(\alpha_{2n-2})\end{matrix} \arrow[l, "\text{interpolation}"']
\end{tikzcd}$$

Polynomial evaluation - a matrix view

$$P(\omega) = \sum_{j=0}^{n-1} a_j \omega_i^j$$

Polynomial evaluation - a matrix view

$$P(\omega) = \sum_{j=0}^{n-1} a_j \omega_i^j$$

$$\begin{pmatrix} P(1) \\ P(\omega) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & (\omega^2)^2 & \cdots & (\omega^2)^{n-1} \\ & \cdots & \ddots & \ddots & \cdots \\ 1 & \omega^{n-1} & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

Polynomial evaluation - a matrix view

$$P(\omega) = \sum_{j=0}^{n-1} a_j \omega_i^j$$

$$\begin{pmatrix} P(1) \\ P(\omega) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & (\omega^2)^2 & \cdots & (\omega^2)^{n-1} \\ & \cdots & \ddots & \ddots & \cdots \\ 1 & \omega^{n-1} & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

Polynomial evaluation - a matrix view

$$P(\omega) = \sum_{j=0}^{n-1} a_j \omega_i^j$$

$$\begin{pmatrix} P(1) \\ P(\omega) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & (\omega^2)^2 & \cdots & (\omega^2)^{n-1} \\ & \cdots & \ddots & \ddots & \cdots \\ 1 & \omega^{n-1} & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

Polynomial evaluation - a matrix view

$$P(\omega) = \sum_{j=0}^{n-1} a_j \omega_i^j$$

$$\begin{pmatrix} P(1) \\ P(\omega) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & (\omega^2)^2 & \cdots & (\omega^2)^{n-1} \\ & \cdots & \ddots & \ddots & \cdots \\ 1 & \omega^{n-1} & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

Polynomial evaluation - a matrix view

$$P(\omega) = \sum_{j=0}^{n-1} a_j \omega_i^j$$

$$\begin{pmatrix} P(1) \\ P(\omega) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & (\omega^2)^2 & \cdots & (\omega^2)^{n-1} \\ & \dots & \ddots & \ddots & \dots \\ 1 & \omega^{n-1} & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

Polynomial evaluation - a matrix view

$$P(\omega) = \sum_{j=0}^{n-1} a_j \omega_i^j$$

$$\begin{pmatrix} P(1) \\ P(\omega) \\ P(\omega^2) \\ \vdots \\ P(\omega^{n-1}) \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & (\omega^2)^2 & \cdots & (\omega^2)^{n-1} \\ & \ldots & \ddots & \ddots & \ldots \\ 1 & \omega^{n-1} & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix}}_{\text{Vandermonde matrix } V(\omega)} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

Polynomial evaluation - a geometric view

Question: What linear transformation does the Vandermonde matrix correspond to?

Polynomial evaluation - a geometric view

Question: What linear transformation does the Vandermonde matrix correspond to?

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & (\omega^2)^2 & \cdots & (\omega^2)^{n-1} \\ & \ldots & \ddots & \ddots & \ldots \\ 1 & \omega^{n-1} & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

Polynomial evaluation - a geometric view

Question: What linear transformation does the Vandermonde matrix correspond to?

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & (\omega^2)^2 & \cdots & (\omega^2)^{n-1} \\ & \dots & \ddots & \ddots & \dots \\ 1 & \omega^{n-1} & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

Polynomial evaluation - a geometric view

Question: What linear transformation does the Vandermonde matrix correspond to?

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & (\omega^2)^2 & \cdots & (\omega^2)^{n-1} \\ & \dots & \ddots & \ddots & \dots \\ 1 & \omega^{n-1} & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

Multiplying the Vandermonde matrix with the i^{th} vector of the standard basis gives the i^{th} column of the Vandermonde matrix

Polynomial evaluation - a geometric view

Lemma: The columns of the Vandermonde matrix $V(\omega)$ are orthogonal vectors

Polynomial evaluation - a geometric view

Lemma: The columns of the Vandermonde matrix $V(\omega)$ are orthogonal vectors

complex conjugate

$$\begin{pmatrix} 1 & \omega^j & (\omega^j)^2 & (\omega^j)^3 & \cdots & (\omega^j)^{n-1} \end{pmatrix} \begin{pmatrix} 1 \\ \omega^{-k} \\ (\omega^{-k})^2 \\ (\omega^{-k})^3 \\ \vdots \\ (\omega^{-k})^{n-1} \end{pmatrix} = \sum_{i=0}^{n-1} \omega^{i(j-k)}$$

Polynomial evaluation - a geometric view

Lemma: The columns of the Vandermonde matrix $V(\omega)$ are orthogonal vectors

complex conjugate

$$\begin{pmatrix} 1 & \omega^j & (\omega^j)^2 & (\omega^j)^3 & \cdots & (\omega^j)^{n-1} \end{pmatrix} \begin{pmatrix} 1 \\ \omega^{-k} \\ (\omega^{-k})^2 \\ (\omega^{-k})^3 \\ \vdots \\ (\omega^{-k})^{n-1} \end{pmatrix} = \sum_{i=0}^{n-1} \omega^{i(j-k)}$$

- If $j = k$, then $\sum_{i=0}^{n-1} \omega^{i(j-k)} = n$
- If $j \neq k$,

$$\sum_{i=0}^{n-1} \omega^{i(j-k)} = \frac{1 - \omega^{n(j-k)}}{1 - \omega^{j-k}}$$

Polynomial evaluation - a geometric view

Lemma: The columns of the Vandermonde matrix $V(\omega)$ are orthogonal vectors

complex conjugate

$$\begin{pmatrix} 1 & \omega^j & (\omega^j)^2 & (\omega^j)^3 & \cdots & (\omega^j)^{n-1} \end{pmatrix} \begin{pmatrix} 1 \\ \omega^{-k} \\ (\omega^{-k})^2 \\ (\omega^{-k})^3 \\ \vdots \\ (\omega^{-k})^{n-1} \end{pmatrix} = \sum_{i=0}^{n-1} \omega^{i(j-k)}$$

- If $j = k$, then $\sum_{i=0}^{n-1} \omega^{i(j-k)} = n$
- If $j \neq k$,

$\omega^n = 1$ - n^{th} root of unity

$$\sum_{i=0}^{n-1} \omega^{i(j-k)} = \frac{1 - \omega^{n(j-k)}}{1 - \omega^{j-k}} = 0$$

Polynomial evaluation - a geometric view

Lemma: The columns of the Vandermonde matrix V(ω) are orthogonal vectors

- The Vandermonde matrix V(ω) rotates the standard basis to the Fourier basis
- The Fourier basis consists of the columns of the Vandermonde matrix V(ω)

Polynomial evaluation - a geometric view

Lemma: The columns of the Vandermonde matrix $V(\omega)$ are orthogonal vectors

- The Vandermonde matrix $V(\omega)$ rotates the standard basis to the Fourier basis
- The Fourier basis consists of the columns of the Vandermonde matrix $V(\omega)$
- Interpolation amounts to performing the reverse transformation

Polynomial evaluation - a geometric view

Lemma: The columns of the Vandermonde matrix $V(\omega)$ are orthogonal vectors

- The Vandermonde matrix $V(\omega)$ rotates the standard basis to the Fourier basis
- The Fourier basis consists of the columns of the Vandermonde matrix $V(\omega)$
- Interpolation amounts to performing the reverse transformation

$$V(\omega)V(\omega)^* = n\mathbf{I}$$

complex conjugate

Interpolation - Inverting the Vandermonde matrix

Interpolation - Inverting the Vandermonde matrix

Lemma: $V(\omega)^{-1} = \frac{1}{n}V(\omega)^*$

$$V(\omega)^* = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & (\omega^{-1})^2 & \cdots & (\omega^{-1})^{n-1} \\ 1 & \omega^{-2} & (\omega^{-2})^2 & \cdots & (\omega^{-2})^{n-1} \\ & \dots & \ddots & \ddots & \dots \\ 1 & \omega^{-(n-1)} & (\omega^{-(n-1)})^2 & \cdots & (\omega^{-(n-1)})^{n-1} \end{pmatrix}$$

Interpolation - Inverting the Vandermonde matrix

Lemma: $V(\omega)^{-1} = \frac{1}{n}V(\omega)^*$

$$V(\omega)^* = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & (\omega^{-1})^2 & \cdots & (\omega^{-1})^{n-1} \\ 1 & \omega^{-2} & (\omega^{-2})^2 & \cdots & (\omega^{-2})^{n-1} \\ & \dots & \ddots & \ddots & \dots \\ 1 & \omega^{-(n-1)} & (\omega^{-(n-1)})^2 & \cdots & (\omega^{-(n-1)})^{n-1} \end{pmatrix}$$

- Conjugates of the roots of unity are themselves roots of unity!

Interpolation - Inverting the Vandermonde matrix

Lemma: $V(\omega)^{-1} = \frac{1}{n}V(\omega)^*$

$$V(\omega)^* = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & (\omega^{-1})^2 & \cdots & (\omega^{-1})^{n-1} \\ 1 & \omega^{-2} & (\omega^{-2})^2 & \cdots & (\omega^{-2})^{n-1} \\ & \dots & \ddots & \ddots & \dots \\ 1 & \omega^{-(n-1)} & (\omega^{-(n-1)})^2 & \cdots & (\omega^{-(n-1)})^{n-1} \end{pmatrix}$$

- Conjugates of the roots of unity are themselves roots of unity!

$\omega^{-j} = \omega^{n-j}$

Interpolation - Inverting the Vandermonde matrix

Lemma: $V(\omega)^{-1} = \frac{1}{n}V(\omega)^*$

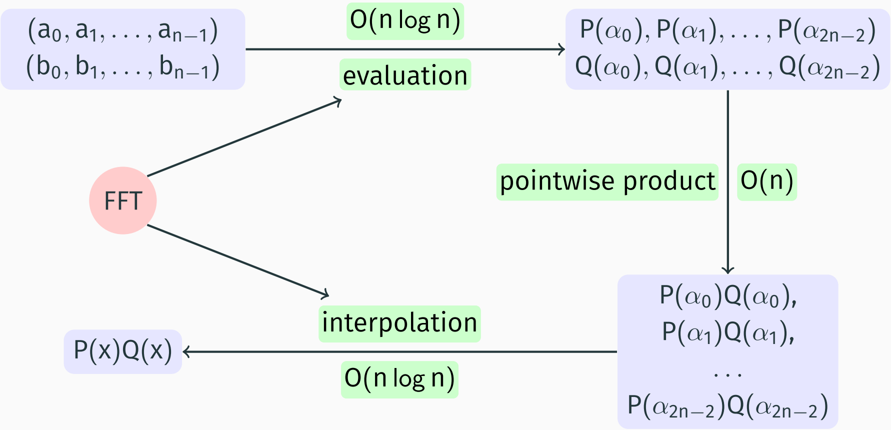
$$V(\omega)^* = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{n-1} & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \\ 1 & \omega^{n-2} & (\omega^{n-2})^2 & \cdots & (\omega^{n-2})^{n-1} \\ & \cdots & \ddots & \ddots & \cdots \\ 1 & \omega & (\omega)^2 & \cdots & (\omega)^{n-1} \end{pmatrix}$$

- Conjugates of the roots of unity are themselves roots of unity!
- Compute the FFT, and reverse the output!

$\omega^{-j} = \omega^{n-j}$

Convolution - the final algorithm

$$P(x) = \sum_{i=0}^{n-1} a_i x^i \quad Q(x) = \sum_{i=0}^{n-1} b_i x^i$$



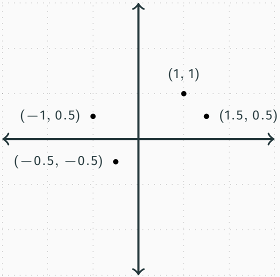
Closest pair of points

Pairwise distances of points

Given n points $\{p_i = (x_i, y_i)\}_{1 \le i \le n}$ on the plane, find a pair of points that are closest to each other

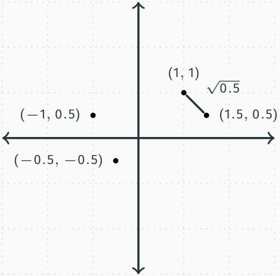
Pairwise distances of points

Given n points $\{p_i = (x_i, y_i)\}_{1 \le i \le n}$ on the plane, find a pair of points that are closest to each other



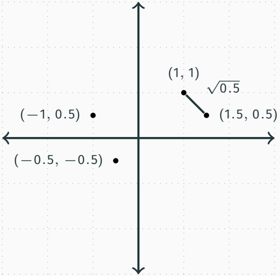
Pairwise distances of points

Given n points $\{p_i = (x_i, y_i)\}_{1 \leq i \leq n}$ on the plane, find a pair of points that are closest to each other



Pairwise distances of points

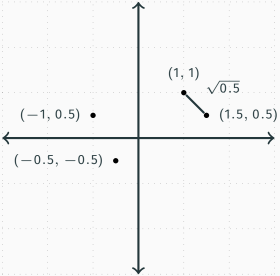
Given n points $\{p_i = (x_i, y_i)\}_{1 \le i \le n}$ on the plane, find a pair of points that are closest to each other



- (Most?) Important primitive for geometric algorithms

Pairwise distances of points

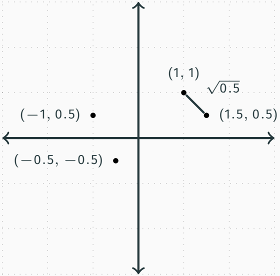
Given n points $\{p_i = (x_i, y_i)\}_{1 \le i \le n}$ on the plane, find a pair of points that are closest to each other



- (Most?) Important primitive for geometric algorithms
- **Question:** What is the simplest algorithm?

Pairwise distances of points

Given n points $\{p_i = (x_i, y_i)\}_{1 \leq i \leq n}$ on the plane, find a pair of points that are closest to each other

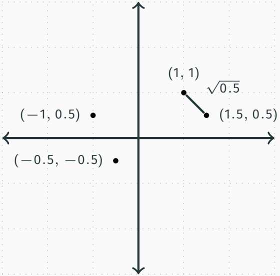


- (Most?) Important primitive for geometric algorithms
- **Question:** What is the simplest algorithm?

  For each pair of points, find the distance; then find the pair that is closest

Pairwise distances of points

Given n points $\{p_i = (x_i, y_i)\}_{1 \le i \le n}$ on the plane, find a pair of points that are closest to each other



- (Most?) Important primitive for geometric algorithms
- **Question:** What is the simplest algorithm?

  For each pair of points, find the distance; then find the pair that is closest → $O(n^2)$

Pairwise distances - the 1D case

Given n points on the real line - find the pair of points that are closest to each other

Pairwise distances - the 1D case

Given n points on the real line - find the pair of points that are closest to each other

- Sort the points
- Find the i such that $p_{i+1} - p_i$ is the smallest in the sorted order

Pairwise distances - the 1D case

Given n points on the real line - find the pair of points that are closest to each other

- Sort the points $\longrightarrow$ $O(n \log n)$
- Find the i such that $p_{i+1} - p_i$ is the smallest in the sorted order
  $\hookrightarrow$ $O(n)$

Pairwise distances - the 1D case

Given n points on the real line - find the pair of points that are closest to each other

- Sort the points $\longrightarrow$ $O(n \log n)$
- Find the i such that $p_{i+1} - p_i$ is the smallest in the sorted order
  $\hookrightarrow$ $O(n)$

Question: Does a similar idea work in the 2D case - sorting based on the x and y coordinates

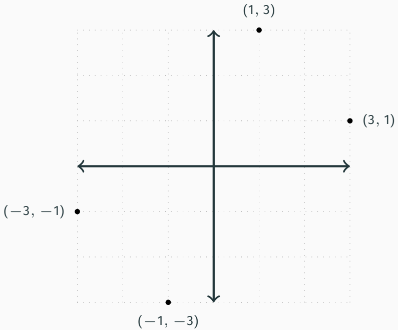
Pairwise distances - the 2D case

Pairwise distances - the 2D case

Sorting according to x-coordinate and y-coordinate and checking in order

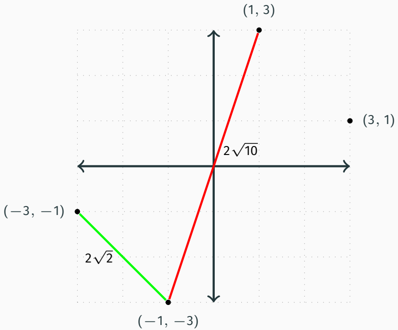
Pairwise distances - the 2D case

Sorting according to x-coordinate and y-coordinate and checking in order



Pairwise distances - the 2D case

Sorting according to x-coordinate and y-coordinate and checking in order

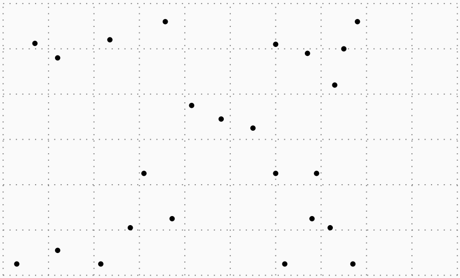


Pairwise distances - a divide and conquer approach

Assumption: No two points have the same x-coordinate

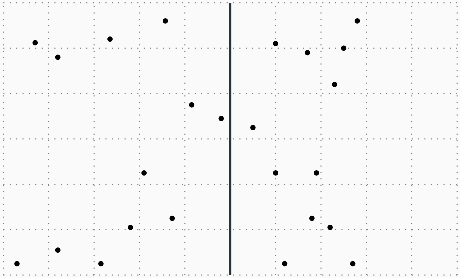
Pairwise distances - a divide and conquer approach

Assumption: No two points have the same x-coordinate



Pairwise distances - a divide and conquer approach

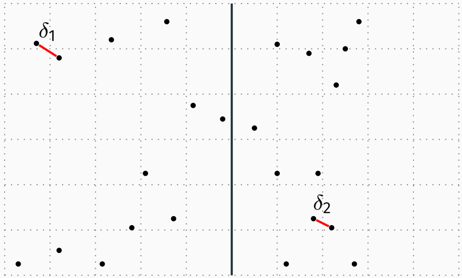
Assumption: No two points have the same x-coordinate



- Divide the set of points into two (almost)equal halves

Pairwise distances - a divide and conquer approach

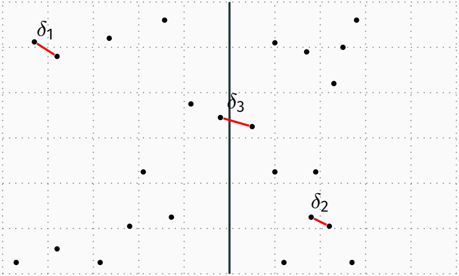
Assumption: No two points have the same x-coordinate



- Divide the set of points into two (almost)equal halves
- Find the closest pair of points in each part recursively

Pairwise distances - a divide and conquer approach

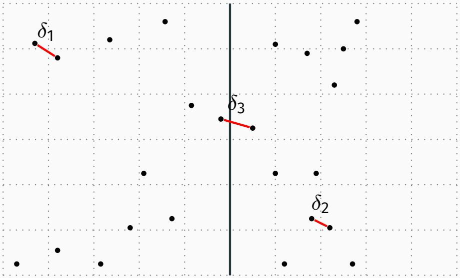
Assumption: No two points have the same x-coordinate



- Divide the set of points into two (almost)equal halves
- Find the closest pair of points in each part recursively
- Find the closest pair of points, one on each side

Pairwise distances - a divide and conquer approach

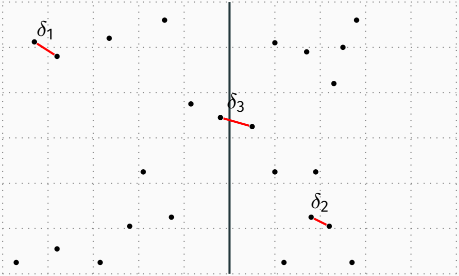
Assumption: No two points have the same x-coordinate



- Divide the set of points into two (almost)equal halves
- Find the closest pair of points in each part recursively
- Find the closest pair of points, one on each side
- Return $\min(\delta_1, \delta_2, \delta_3)$

Pairwise distances - a divide and conquer approach

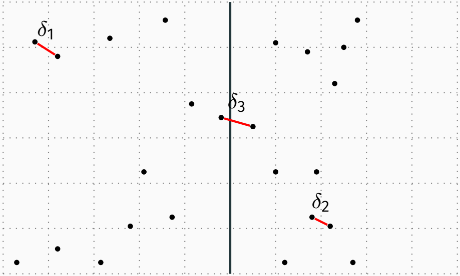
Assumption: No two points have the same x-coordinate



- Divide the set of points into two (almost)equal halves
- Find the closest pair of points in each part recursively — $2T(n/2)$
- Find the closest pair of points, one on each side
- Return $\min(\delta_1, \delta_2, \delta_3)$

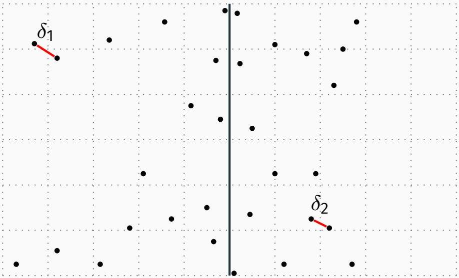
Pairwise distances - a divide and conquer approach

Assumption: No two points have the same x-coordinate

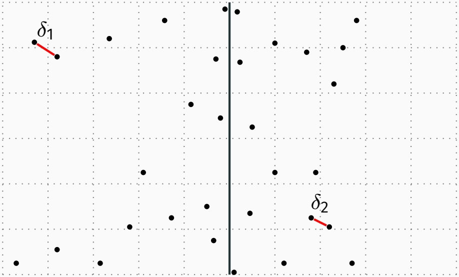


- Divide the set of points into two (almost)equal halves
- Find the closest pair of points in each part recursively $2T(n/2)$
- Find the closest pair of points, one on each side $O(n^2)$?
- Return $\min(\delta_1, \delta_2, \delta_3)$

Pairwise distances - a divide and conquer approach

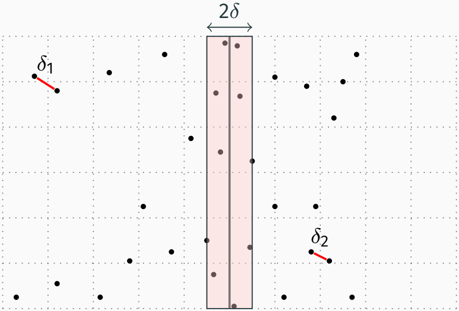


Pairwise distances - a divide and conquer approach



$\delta = \min(\delta_1, \delta_2)$

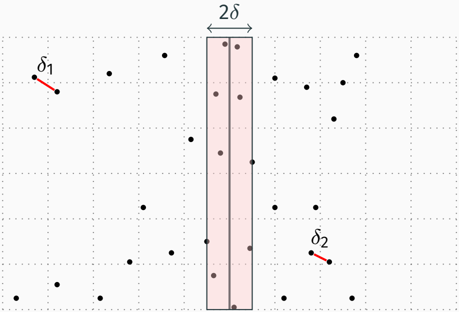
Pairwise distances - a divide and conquer approach



$\delta = \min(\delta_1, \delta_2)$

- Only points within δ of the midpoint line need to be considered

Pairwise distances - a divide and conquer approach

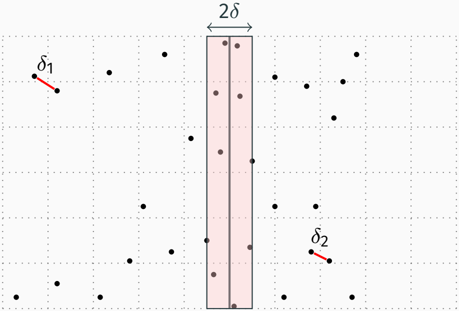


$\delta = \min(\delta_1, \delta_2)$

- Only points within δ of the midpoint line need to be considered

  might contain all the points!

Pairwise distances - a divide and conquer approach



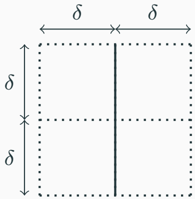
$\delta = \min(\delta_1, \delta_2)$

- Only points within δ of the midpoint line need to be considered

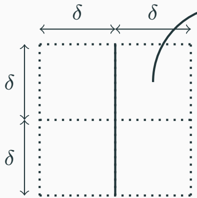
    might contain all the points!

Question: For a fixed point p on one side, how many candidate points are there on the other side?

Pairwise distances - bounding the number of points

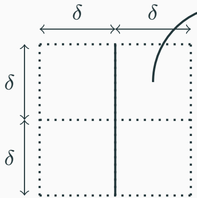


Pairwise distances - bounding the number of points



How many points can lie inside the $\delta \times \delta$ square?

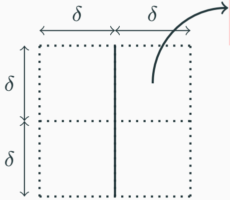
Pairwise distances - bounding the number of points



How many points can lie inside the $\delta \times \delta$ square?

- Every two points inside a $\delta \times \delta$ square must be at least δ apart!

Pairwise distances - bounding the number of points

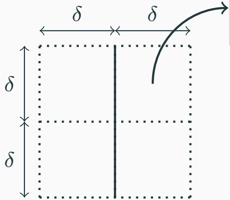


How many points can lie inside the $\delta \times \delta$ square?

- Every two points inside a $\delta \times \delta$ square must be at least δ apart!

there can be no more than 4 points inside a $\delta \times \delta$ square

Pairwise distances - bounding the number of points



How many points can lie inside the $\delta \times \delta$ square?

- Every two points inside a $\delta \times \delta$ square must be at least δ apart!

  there can be no more than 4 points inside a $\delta \times \delta$ square

- For any point p in the 2δ strip, need to consider at most 15 points next to it, when the points are sorted according to the y coordinate

Closest pair of points - a divide and conquer algorithm

Input: Set of points $p_i = (x_i, y_i)$
Output: Pair of points closest to each other
Find the line $x = x_m$ dividing into two equal halves P_1 and P_2
$\delta_1 \leftarrow$ distance between closest pair of points in P_1
$\delta_2 \leftarrow$ distance between closes pair of points in P_2
$\delta \leftarrow \min(\delta_1, \delta_2)$
Sort the points P_δ between $x = x_m - \delta$ and $x_m + \delta$ according to the y coordinates
foreach point p in the sorted list of P_δ **do**
    foreach point p' that is at most 15 positions away from p **do**
        $\delta' \leftarrow \min(\delta', d(p, p'))$
return $\min(\delta', \delta)$

Closest pair of points - a divide and conquer algorithm

Input: Set of points $p_i = (x_i, y_i)$
Output: Pair of points closest to each other
Find the line $x = x_m$ dividing into two equal halves P_1 and P_2 $\longrightarrow$ $O(n)$
$\delta_1 \leftarrow$ distance between closest pair of points in P_1
$\delta_2 \leftarrow$ distance between closes pair of points in P_2 $\longrightarrow$ $2T(\frac{n}{2})$
$\delta \leftarrow \min(\delta_1, \delta_2)$
Sort the points P_δ between $x = x_m - \delta$ and $x_m + \delta$ according to the y coordinates $\longrightarrow$ $O(n \log n)$
foreach point p in the sorted list of P_δ **do** $\longrightarrow$ $O(n)$
    foreach point p' that is at most 15 positions away from p **do**
        $\delta' \leftarrow \min(\delta', d(p, p'))$
return $\min(\delta', \delta)$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n)$$

Closest pair of points - solving the recurrence

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n)$$

Closest pair of points - solving the recurrence

$$\underbrace{T(n) = 2T\left(\tfrac{n}{2}\right) + O(n \log n)}$$

$$a \quad b \quad f(n)$$

Recall the formula from the recursion tree:
$T(n) = n^{\log_b a} + \sum_{i=0}^{\log n - 1} a^i f(n/b^i)$

Closest pair of points - solving the recurrence

$$\boxed{T(n) = \underset{\substack{\downarrow \\ a}}{2}T\Big(\underset{\substack{\downarrow \\ b}}{\frac{n}{2}}\Big) + O(\underset{\substack{\downarrow \\ f(n)}}{n \log n})}$$

Recall the formula from the recursion tree:
$T(n) = n^{\log_b a} + \sum_{i=0}^{\log n - 1} a^i f(n/b^i)$

$$T(n) = n + \sum_{i=0}^{\log n - 1} 2^i \frac{n}{2^i} \log\left(\frac{n}{2^i}\right)$$

$$= n + n \log\left(\prod_{i=0}^{\log n - 1} \frac{n}{2^i}\right) = n + n \log\left(\frac{n^{\log n}}{2^{\sum_{i=0}^{\log n - 1} i}}\right)$$

$$= n + n \log\left(\frac{n^{\log n}}{2^{\log n (\log n - 1)/2}}\right) = O(n \log^2 n)$$

Closest pair of points - solving the recurrence

$$\underbrace{T(n) = \underset{\downarrow}{2}T\left(\underset{\downarrow}{\tfrac{n}{2}}\right) + \underset{\downarrow}{O(n \log n)}}_{}$$

a b $f(n)$

$\longrightarrow$ Can we avoid sorting at each step?

Recall the formula from the recursion tree:
$T(n) = n^{\log_b a} + \sum_{i=0}^{\log n - 1} a^i f(n/b^i)$

$$T(n) = n + \sum_{i=0}^{\log n - 1} 2^i \frac{n}{2^i} \log\left(\frac{n}{2^i}\right)$$

$$= n + n \log\left(\prod_{i=0}^{\log n - 1} \frac{n}{2^i}\right) = n + n \log\left(\frac{n^{\log n}}{2^{\sum_{i=0}^{\log n - 1} i}}\right)$$

$$= n + n \log\left(\frac{n^{\log n}}{2^{\log n(\log n - 1)/2}}\right) = O(n \log^2 n)$$

Closest pair of points - preprocessing the input

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n) \longrightarrow$$

Can we avoid sorting at each step?

Closest pair of points - preprocessing the input

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n) \longrightarrow$$

Can we avoid sorting at each step?

- Sort the list of points P according to both the x and y coordinates and maintain two lists P_x and P_y

Closest pair of points - preprocessing the input

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n) \longrightarrow$$

Can we avoid sorting at each step?

- Sort the list of points P according to both the x and y coordinates and maintain two lists P_x and P_y
- At each recursive call, divide the the list into two lists maintaining the sorted order

Closest pair of points - preprocessing the input

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n) \longrightarrow$$

Can we avoid sorting at each step?

- Sort the list of points P according to both the x and y coordinates and maintain two lists P_x and P_y
- At each recursive call, divide the the list into two lists maintaining the sorted order
- Scan the sorted list to obtain the list of points in the 2δ-strip sorted according to y coordinates

Closest pair of points - preprocessing the input

$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n)$ $\longrightarrow$ Can we avoid sorting at each step?

- Sort the list of points P according to both the x and y coordinates and maintain two lists P_x and P_y
- At each recursive call, divide the the list into two lists maintaining the sorted order $\longrightarrow$ $O(n)$
- Scan the sorted list to obtain the list of points in the 2δ-strip sorted according to y coordinates $\longrightarrow$ $O(n)$

Closest pair of points - preprocessing the input

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log n) \longrightarrow$$ Can we avoid sorting at each step?

- Sort the list of points P according to both the x and y coordinates and maintain two lists P_x and P_y
- At each recursive call, divide the the list into two lists maintaining the sorted order $\longrightarrow O(n)$
- Scan the sorted list to obtain the list of points in the 2δ-strip sorted according to y coordinates $\longrightarrow O(n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \text{ + one-time cost of sorting}$$