# CSC236 *Intro. to* the Theory of Computation

# Lecture 7: Master Theorem; more D&C; correctness

Amir H. Chinaei, Fall 2016

Office Hours: W 2-4 BA4222

ahchinaei@cs.toronto.edu
*http://www.cs.toronto.edu/~ahchinaei/*

*Course page:*
*http://www.cdf.toronto.edu/~csc236h/fall/index.html*

*Section page:*
*http://www.cdf.toronto.edu/~csc236h/fall/amir_lectures.html*

# review

- Last week
  - application of recurrence relations to complexity of d&c algorithms
    - in particular *mergeSort*
- this week
  - master theorem
  - *closestPair*
  - recursive correctness

Keep in mind: $T(\frac{\hat{n}}{2}) \leq T(n) \leq T(\hat{n})$

and $T(\hat{n}) = \hat{n}\log\hat{n} + 2\hat{n} - 1$

# Example 63: *mergeSort*

- calculating a lower bound

  $T(n) \geq c\, n \log n$

$$
\begin{aligned}
T(n) \geq T\left(\frac{\hat{n}}{2}\right) & \\
&= \frac{\hat{n}}{2}\log\frac{\hat{n}}{2} + 2\frac{\hat{n}}{2} - 1 \\
&= \frac{\hat{n}}{2}(\log\hat{n} - \log 2) + \hat{n} - 1 \\
&= \frac{\hat{n}}{2}\log\hat{n} + \frac{\hat{n}}{2} - 1 \\
&\geq \frac{n}{2}\log n + \frac{n}{2} - 1 \\
&\geq \frac{n}{2}\log n
\end{aligned}
$$

$$c = \frac{1}{2} \quad n \geq 2$$

□

Keep in mind: $T(\frac{\hat{n}}{2}) \leq T(n) \leq T(\hat{n})$

and $T(\hat{n}) = \hat{n}\log\hat{n} + 2\hat{n} - 1$

# Example 63: *mergeSort*

- calculating an upper bound

$$T(n) \leq c\, n \log n$$

$$\begin{aligned}
T(n) &\leq T(\hat{n}) \\
&= \hat{n}\log\hat{n} + 2\hat{n} - 1 \\
&\leq 2n\log 2n + 2.2n - 1 \\
&= 2n(\log 2 + \log n) + 4n - 1 \\
&= 2n\log n + 6n - 1 \\
&\leq 2n\log n + 6n \\
&\leq 2n\log n + 6n\log n \\
&\leq 8n\log n
\end{aligned}$$

$$c = 8 \quad n \geq 2$$

□

# general d&c and master theorem

- D&C algorithms normally divide a problem of size $\boldsymbol{n}$ to $\boldsymbol{a}$ smaller problems of size $\frac{n}{\boldsymbol{b}}$ where $a > 0\,, b > 1 \quad \in \mathbb{N}$
- Let $g(x)$ denote the recombining cost (conquer), such that the corresponding recurrence relation is

$$T(n) = \begin{cases} 1 & n \leq B \in \mathbb{N} \\ aT\left(\frac{n}{b}\right) + g(x) & n > B \in \mathbb{N} \end{cases}$$

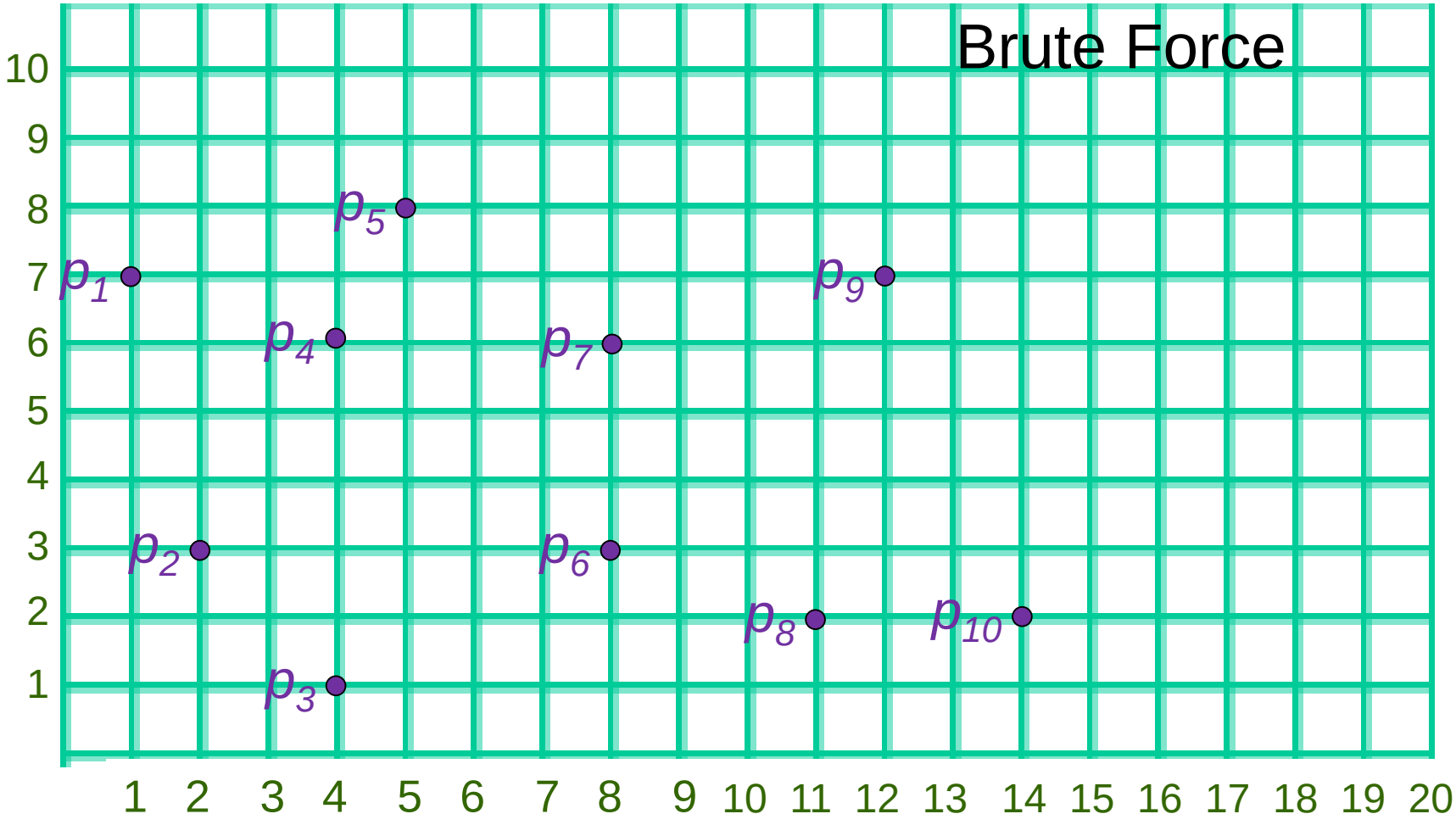
- When $g(x) = \boldsymbol{c}n^{\boldsymbol{d}},\ c > 0\ d \geq 0 \in \mathbb{R}$

$$T(n) \in \begin{cases} \theta\left(n^d\right) & if\ a < b^d \\ \theta\left(n^d \log n\right) & if\ a = b^d \\ \theta\left(n^{\log_b a}\right) & if\ a > b^d \end{cases}$$

# notes:

$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

# Example 64: *closestPair*




min=

$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

# Example 64: *closestPair*

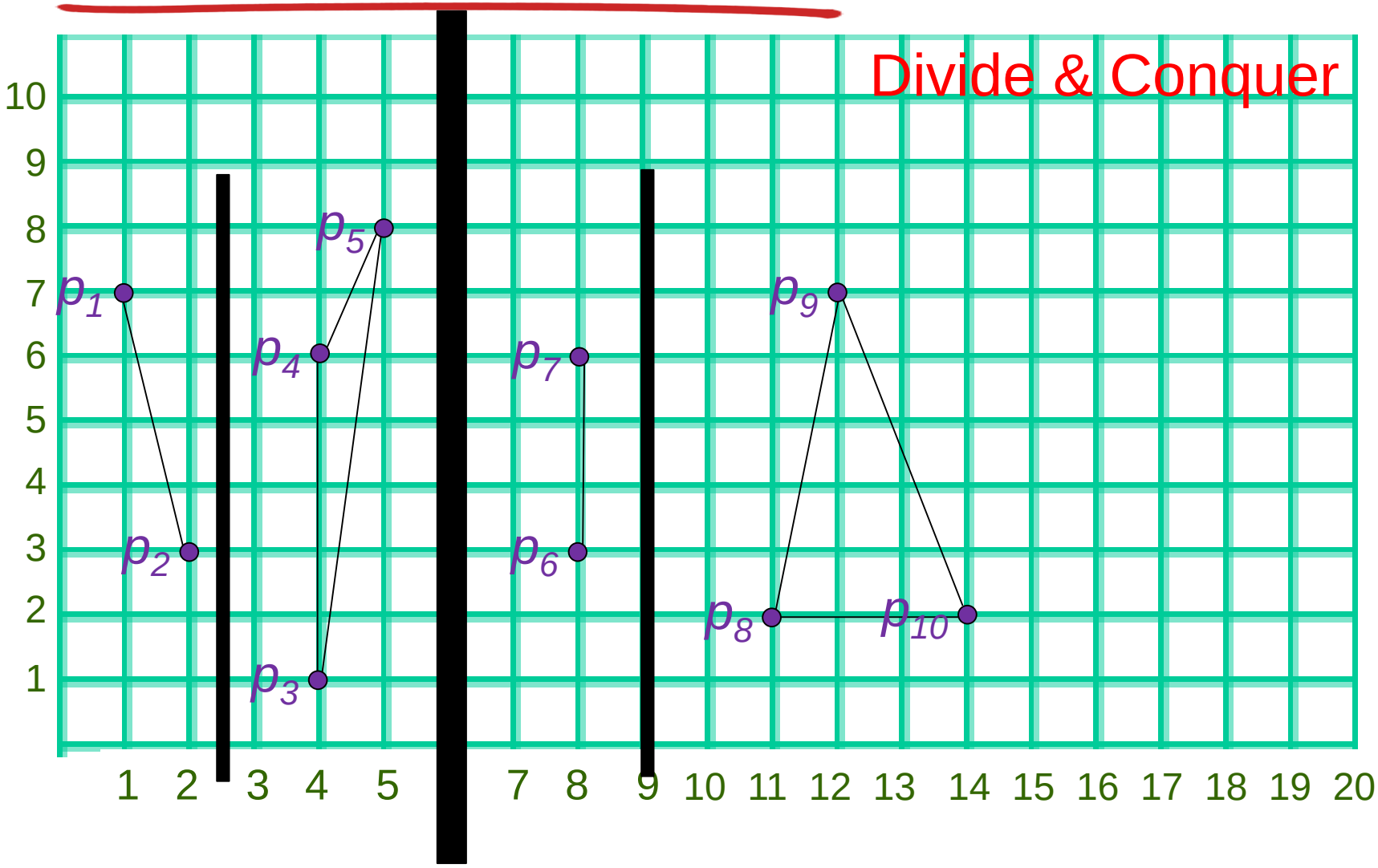
Brute Force

```
s = [set of points]

def closest_pair1(frm,to):
    min_d = d(s[0],s[1])
    for i in range(to-frm):
        for j in range(i+1,to-frm+1):
            dist = d(s[i],s[j])
            if dist < min_d:
                min_d = dist
    return min_d
```

$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

# Example 64: *closestPair*



$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

# Example 64: *closestPair*

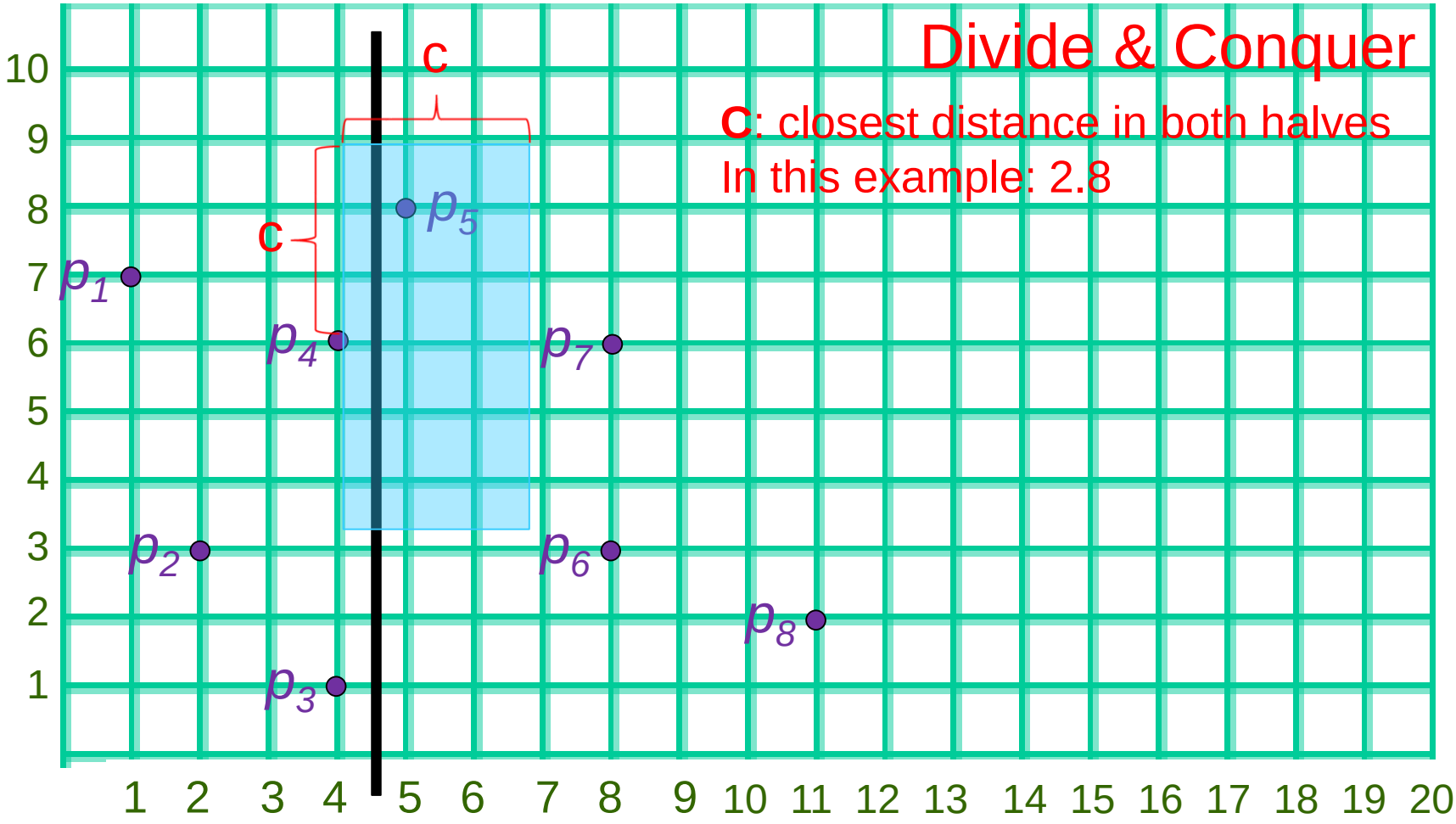
```
s = [set of points]

def closest_pairDC0(frm, to):
    if to-frm > 1:
        mid = (frm+to)//2
        half1 = closest_pairDC0(frm, mid)
        half2 = closest_pairDC0(mid+1, to)
        c = min(half1,half2)
        border_min = border_bf(mid, c)
        return min(c, border_min)
    else:
        return d(s[frm],s[to])
```

# analysis:

$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

# Example 64: *closestPair*



$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

# Example 64: *closestPair*

```
s = [set of points, sorted by x-coordinates]

def closest_pairDC1(frm, to):
    if to-frm > 1:
        mid = (frm+to)//2
        half1 = closest_pairDC1(frm, mid)
        half2 = closest_pairDC1(mid+1, to)
        c = min(half1,half2)
        mergeSort(s, on y-coordinates)
        border_min = border_n(mid, c)
        return min(closest, border_min)
    else:
        return d(s[frm],s[to])
```

# analysis:

$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

# Example 64: *closestPair*

```
s = [set of points, sorted by x-coordinates]

def closest_pairDC(frm, to):
    if to-frm > 1:
        mid = (frm+to)//2
        half1 = closest_pairDC(frm, mid)
        half2 = closest_pairDC(mid+1, to)
        c = min(half1,half2)
        merge(half1, half2, y-coordinates)
        border_min = border_n(mid, c)
        return min(closest, border_min)
    else:
        return d(s[frm],s[to])
```

# analysis:

# programs correctness

- Recently, we saw the application of *induction* in **asymptotic analysis**
  - in particular, in the worst case time complexity of recursive algorithms

- Let's move on to see application of *induction* in a more important topic: **correctness** of recursive algorithms
  - followed by correctness of iterative algorithms, next week

# *Example 73: correctness of binSearch* (from Example 61)

```
def binSearch(x, A, b, e):
  1   if b == e:
  2       if x == A[b]:
  3           return b
  4       else:
  5           return -1
  6   else:
  7       m = (b + e) // 2          # midpoint
  8       if x <= A[m]:
  9           return binSearch(x, A, b, m)
  10      else:
  11          return binSearch(x, A, m+1, e)
```

# how to devise the correctness proof:

- define the pre- & post- conditions for the algorithm
  - preconditions:
    - conditions that the algorithm's input should satisfy
  - postconditions:
    - conditions that should be satisfied after the algorithm has run
- then, show:

$$preconditions \Rightarrow postconditions$$

# *Example 73: correctness of binSearch:*

- `binSearch(x, A, b, e)`:
  - **preconditions:**
    - elements of $A$ (from $b$ to $e$) are sorted non-decreasingly
    - elements of A and x are comparable
    - $0 \leq b \leq e$
    - $Len(A) = n = e - b + 1$
  - **postconditions:**
    - `binSearch(x, A, b, e)` terminates and returns $p$ such that $b \leq p \leq e$ and $x = A[p]$ if such a $p$ exists; otherwise returns $-1$.

# *Example 73: correctness of binSearch:*

- We want to show: preconditions $\Rightarrow$ postconditions

  $\boldsymbol{P(n)}$: if $0 \leq b \leq e$ where A$[b..e]$ is non-decreasing and $\text{Len}(\text{A}) = n = e - b + 1$ , and x is comparable to elements of A, the call to `binSearch(x,A,b,e)` terminates and returns $p$ such that $b \leq p \leq e$ and $x = A[p]$ if such a $p$ exists; otherwise returns $-1$.

- Proof by complete induction.
  - **Basis step**:

  - **Inductive step**:

# proof by induction:

# proof by induction:

□

# notes: