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

Example 63: *mergeSort*

Keep in mind: $T(\frac{\hat{n}}{2}) \le T(n) \le T(\hat{n})$
and $T(\hat{n}) = \hat{n}\log\hat{n} + 2\hat{n} - 1$

- calculating a lower bound
  $T(n) \ge c\, n \log n$

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

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

□

Example 63: *mergeSort*

Keep in mind: $T(\frac{\hat{n}}{2}) \le T(n) \le T(\hat{n})$
and $T(\hat{n}) = \hat{n}\log\hat{n} + 2\hat{n} - 1$

- calculating an upper bound
  $T(n) \le c\, n \log n$

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

$c = 8 \quad n \ge 2$

□

general d&c and master theorem

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

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

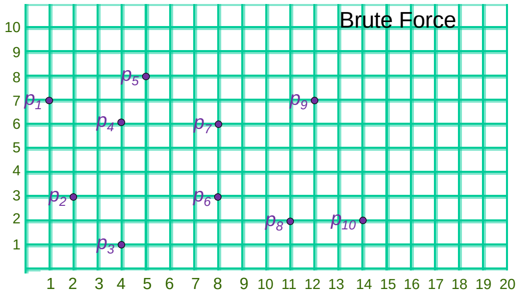
- When $g(x) = cn^d$, $c > 0\ d \ge 0 \in \mathbb{R}$

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

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*

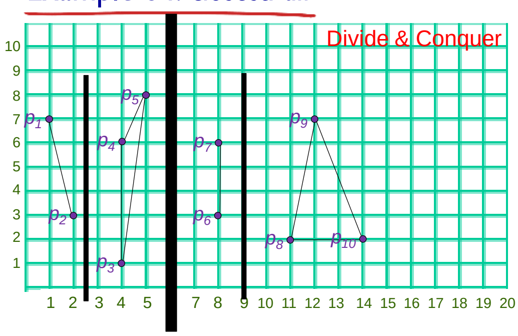
Brute Force

```
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*

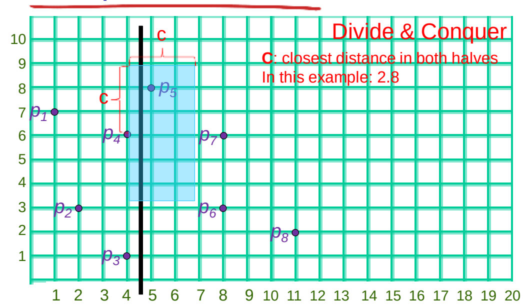
```
s = [set of points]

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):
    if b == e:
        if x == A[b]:
            return b
        else:
            return -1
    else:
        m = (b + e) // 2          # midpoint
        if x <= A[m]:
            return binSearch(x, A, b, m)
        else:
            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 $\mathrm{Len(A)} = n = e - b + 1$, 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: