Lower Bounds by Reduction

- Suppose we already have a lower bound $S(n)$ for some problem A and a reduction from A to B
  - A reduction is a function f such that $A(x) = B(f(x))$
- Let $T_f(n)$ be an upper bound on the time to compute $f(x)$ when $|x| = n$
- Let $T_B(n)$ be an upper bound on the time to compute $B(x)$ when $|x| = n$
- Then computing $A(x)$ as $B(f(x))$ takes at most $T_f(n) + T_B(T_f(n))$

Lower Bound

Since: $S(n) \leq T_f(n) + T_B(T_f(n))$

So: $T_B(n) \geq T_f^{-1}(S(n) - T_f(n))$

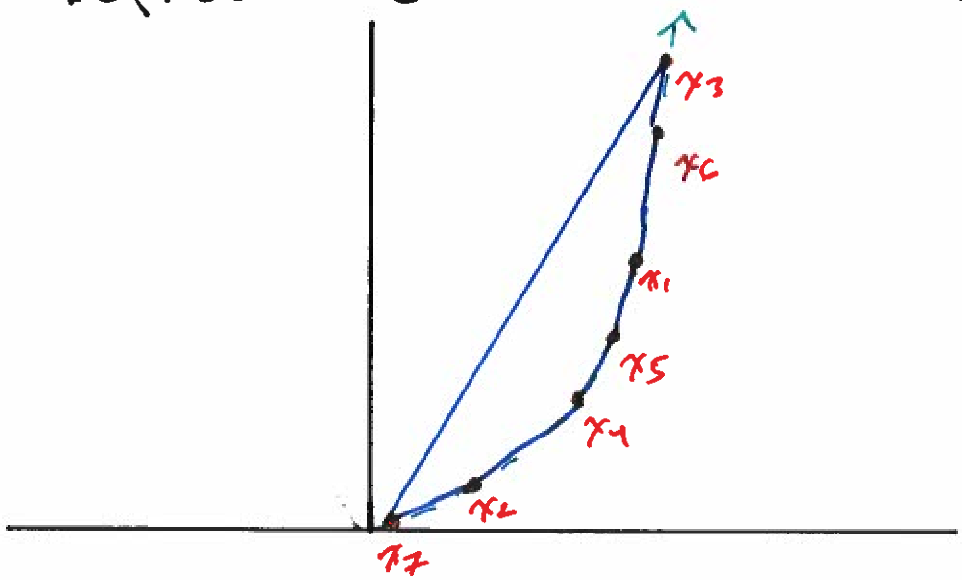
- Consider a reduction from sorting to convex hull
  - the reduction take $O(n)$-time

$$T_B(n) \in \Omega(n \log n) - O(n)$$

$$\in \Omega(n \log n)$$

Sorting to Convex Hull

- To reduce sorting to convex hull
  - assume each element to be sorted is a real number ≥ 0
- Represent each x_i as the point (x_i, x_i^2)



Closest Pair Lower Bound

- To establish a lower bound on closest pair we reduce Element Uniqueness to closest pair.
- Associate each x_i with the point

  $p_i = (x_i, 0)$
- The x_i are unique iff the closest pair of points p_i, p_j are a non-zero distance apart.
- Therefore if we could solve closest pair in $o(n \log n)$-time we could also solve Element Uniqueness in $o(n \log n)$.

Line Segment Intersection Lower Bound

- We can reduce Element Uniqueness to 1D segment intersection
- Represent each element x_i as an interval $[x_i, x_i]$
- The elements are unique iff the intervals don't overlap.

Median Finding

- Given an array of size n, containg integers in some arbitrary order, find the median element.
  - assume n is odd
  - assume all elements are distinct
- The median is the d^{th} smallest element where $d = \lfloor \frac{n}{2} \rfloor + 1$

Brute Force

- compare each x_i to all other elements to find, c, the number of elements smaller than x_i
- Worst case: $n(n-1)$ comparisons $\in \Theta(n^2)$

Reduction

- suppose A is sorted in increasing order

  $A[d]$ is the median

- we can find the median by first sorting the list

  $\Theta(n \log n)$

- Can we do better? yes

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

(CLRS p.171)

- choose an arbitrary pivot
- partition the array into subarrays
  - all items less than the pivot
  - all items greater than the pivot
- recursively sort the subarrays

Complexity

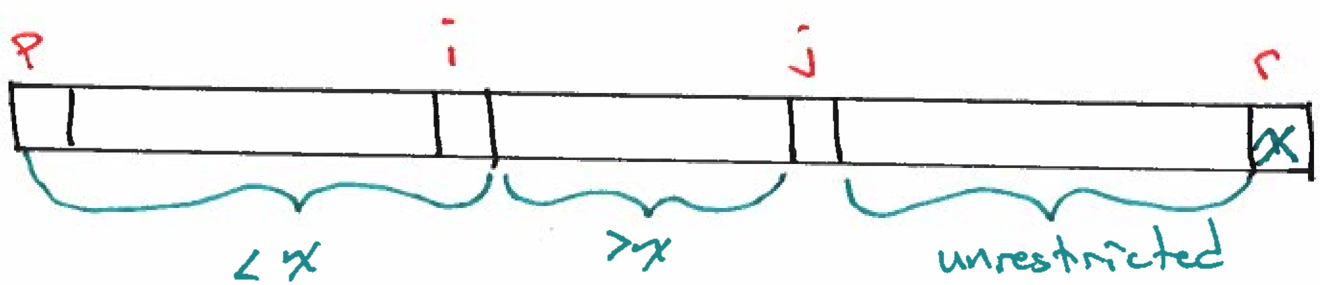
$n = r - p + 1$

$\Theta(n)$-time

PARTITION(A, p, r)

```
x = A[r]
i = p - 1
for j = p to r - 1
        if A[j] ≤ x
                i = i + 1
                exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

(CLRS p.171)



```
QuickSelect(A, p, r, i)
if p == r
    return A[p]
q = Partition(A, p, r) // A[r] is the pivot value
k = q − p + 1 // k is the size of the lower partition
if i == k
    return A[q]
elseif i < k
    return Quick-Select(A, p, q − 1, i)
else
    return Quick-Select(A, q + 1, r, i − k)
```

Best Case:

First pivot is the median

$\Theta(n)$

Worst-case:

$\Theta(n^2)$

- unlucky pivot choice

Quick Select Worst-Case

- Let ~~T(p,r)~~ $T(p,r)$ be the worst case running time of Quick Select on the subarray $A[p \ldots r]$
- Two components contribute to the running time
    - call to partition $\Theta(r-p+1)$-time
    - the recursive call
        - assume the larger partition

$$T(p,r) = \Theta(r-p+1) + T\left(\max_{q \in [p,r]} (q-p),(r-q)\right)$$

- let $n = r-p+1$

$$T(n) = \Theta(n) + T(n-1)$$

- pick a constant to approximate the linear term

$$T(n) = an + T(n-1)$$

- Expand this out (take $T(0) = 0$)

$$T(n) = an + a(n-1) + a(n-2) + \dots + 0$$

$$= \sum_{i=0}^{n} ai$$

$$= a \sum_{i=0}^{n} i$$

$$= a \frac{n(n+1)}{2} \in \Theta(n^2)$$

Linear Time Selection

- If we can improve the pivot choice so that $\max(q-p, r-q) \leq c(r-p+1)$ for some c where $0 < c < 1$

$$T(n) = an + T(cn)$$

- Expand for L levels

$$T(n) = an + acn + ac^2n + ac^3n$$

$$= an \sum_{i=0}^{L} c^i \leq an \sum_{i=0}^{\infty} c^i$$

$$\leq an\left(\frac{1}{1-c}\right) \in \Theta(n)$$