Survey common running times
When do we see running times: \( O(n) \), \( O(n \log n) \), \( O(n^2) \), \( O(\log n) \)

Linear time running time \( \leq C \cdot n \) size of input
- one pass to input data.

1) Finding max \( n \) numbers:

```plaintext
input: sequence \( a_1, a_2, \ldots, a_n \)
max = \( a_1 \)
for \( i = 2 \) to \( n \)
    if \( a_i \geq \text{max} \) then
        max = \( a_i \)
return max.
```

2) Merging two sorted lists

```plaintext
input: list \( a_1, a_2, \ldots, a_n \) sorted
    \( b_1, b_2, \ldots, b_m \) sorted
goal: return list \( c_1, c_2, \ldots, c_m+n \) of \( a_i, b_i \) sorted
ex: 1 3 5 9 \( \rightarrow \) 1 2 3 5 8 9
    2 4 6 8
```

Option 1: put one list next to the other and sort
```
1 3 5 9 2 4 6 8 1 2 3 5 4 9 6 8
1 3 5 2 9 4 6 8 1 2 3 4 5 9 6 8 wasteful
1 3 2 5 9 4 6 8 1 2 3 4 5 6 9 8 (what is the running time of this?)
1 2 3 5 9 4 6 8 1 2 3 4 5 6 8 9
```
option 2  arrange two lists in a stack (as if they were cards and compare the top card in each stack. and append smallest to new list

merge sorted lists

pointer on first element of each list \text{Current}(a), \text{Current}(b)

while both lists are not empty

\text{ai, bj} \text{ elements pointed by } \text{Current}(a), \text{Current}(b)

append min (ai, bj) to output list

advance pointer of list where min was selected

end while

If one list is empty append rest of other list two output

Why option 2 is linear?

- each iteration of while loop one element is added to output so there are at most 2n iterations (\leq 2n when one list is exhausted before the other), each iteration involves constant work.

Note: running time of option 1 is at most

\[ n + \binom{n}{2} + (n-2) + ... + 2 + 1 \] comparisons.

\[ \frac{n(n+1)}{2} \]
**O(n \log n) time**

- Running time of algorithm that splits input into 2 equal pieces, applies algorithm to each piece, and combines two solutions in linear time.

  - **Mergesort algorithm for sorting a list.**
    
    ![Mergesort Diagram](image)
    
    - Sorts 1st piece
    - Sorts 2nd piece
    - Merge sorted lists (linear)

**Quadratic time O(n^2)**

1) Brute-force search of 2 closest points among n points \([x_i, y_i]\) in the plane.

![Closest Points Diagram](image)

- Need to check all pairs of points, \( \binom{n}{2} = \frac{n(n-1)}{2} \) such pairs.
  - Pick pair with min distance. \((x_i, y_i) \text{  (x_j, y_j)}\)
    
    
    \[
    \text{distance} = \sqrt{(x_j-x_i)^2 + (y_j-y_i)^2}
    \]

    (this calculation takes constant work)

    Running time is \(O(n^2)\)

**Note**: There is an \(O(n \log n)\) algorithm to solve this problem later in Dynamic Programming.

2) Two nested loops: one loop of \(n\) iterations, inside loop another loop with \(n\) iterations.

   - For each point \((x_i, y_i)\)
     - For each other point \((x_j, y_j)\)
       - Calculate distance: \(d = \sqrt{(x_j-x_i)^2 + (y_j-y_i)^2}\)
       - If \(d < \text{min recorded distance}\)
         - Update \(\text{min} = d\)

   and for

   end for
Cubic time $O(n^3)$: three nested loops.

1) Set $S_1, S_2, \ldots, S_n$ each subset of $\{1, 2, \ldots, n\}$

find pair of disjoint subsets

for each pair $(S_i, S_j) \leftarrow \binom{n}{2}$ iteration

"check if $S_i$ and $S_j$ have element in common."

for each $p$ in $S_i$

check if $p$ is also in $S_j$

end for

if no element is in $S_i$ is in $S_j$

return $(S_i, S_j)$ disjoint

end if

end for

This is $O\left(\binom{n}{2} \cdot n\right) = O(n^3)$.

$O(n^k)$ time since

$$\binom{n}{k} = \frac{n(n-1)(n-2) \cdots (n-k+1)}{k!} = O(n^k)$$

$= \# k$-element subsets of $\{1, 2, \ldots, n\}$

Brute force search over $k$-element subsets of an $n$ element set

$O(2^n)$ time since $2^n = \# \text{subsets of } \{1, 2, \ldots, n\}$

Brute force search over all subsets of $\{1, 2, \ldots, n\}$

$O(n!)$ time

$n! = \# \text{permutations of } \{1, 2, \ldots, n\}$

(i.e. $\# \text{bijection } \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, n\}$)

Brute force search over all permutations of $\{1, 2, \ldots, n\}$

$2^n \ll n!$
Sublinear time

Since reading input size \( n \) takes \( n \) steps, sublinear running time happens when we can query input indirectly. Goal is to do \( \ll n \) queries.

1) Binary search of a sorted list.

- **Input**: Sorted array \( A \), length \( n \).
- **Goal**: Check if \( p \) is in array.

```
BinarySearch(A)
```

i) Query middle entry \( q \) of \( A \)

- if \( q = p \) return true
- if \( q > p \) then BinarySearch(1st half of \( A \))
- if \( q < p \) then BinarySearch(2nd half of \( A \))

* At each step we chop the list where \( p \) is by a factor of \( \frac{1}{2} \)
* After \( k \) steps we are looking at list of size \( \leq \left( \frac{1}{2} \right)^k n \)
* Number of iterations: \( k \) where \( \left( \frac{1}{2} \right)^k = O \left( \frac{1}{n} \right) \)

i.e. \( k = \log_2 n \)

* Running time at most \( \log_2 n = O(\log n) \) iterations.