Brute Force Solution

No effort to reduce steps, just use a more powerful computer to run faster!

➪ Rely on computer power

- ✎ Directly apply definition
- ✎ Do all possible steps
- ✎ Try all possible alternatives

➪ Think brute force

Exponentiation, matrix multiplication, sorting, string matching, search (next)

Brute Force
Selection Sort

➪ **Key comparison**

Algorithm *SelectionSort*
Input ··· $A[0..n-1]$ ···
Output ···

```
1: for i ← 0 to n − 2 do
2:     min ← i
3:     for j ← i + 1 to n − 1 do
4:         if A[j] < A[min] then
5:             min ← j
6:     swap A[i], A[min]
```

Limits

Use a small instance to figure out/design steps before writing a pseudo-code ***(mix natural language, math, and programming-like expressions)***.

89	45	68	90	29	34	17
17	45	68	90	29	34	89
17	29	68	90	45	34	89
17	29	34	90	45	68	89
17	29	34	45	90	68	89
17	29	34	45	68	90	89
17	29	34	45	68	89	90

➪ **Operation**

➪ **Efficiency?**

Quiz
What would be a <u>suitable</u> basic operation? Can we pick + or – in line 3? What about loop exit check or index increment?

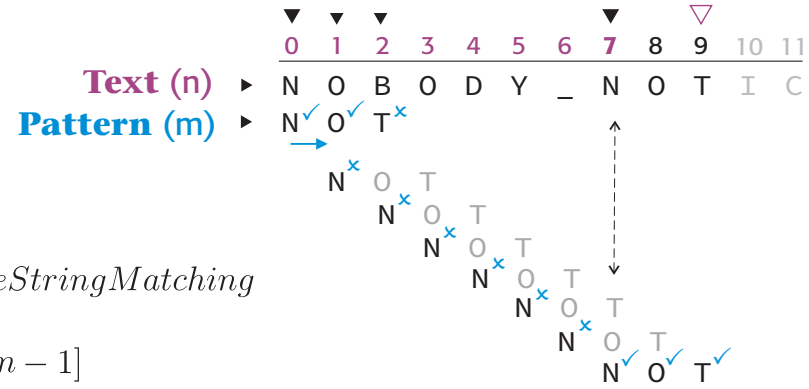
❶ Select input size parameter *n*
❷ Identify basic operation
❸ Check basic op count dependancy
❹ Setup a sum or a recurrence for $C(n)$
❺ Determine order of growth of $C(n)$ (may need to solve sum or recurrence)

Brute Force String Matching

➪ **Text, pattern**

Approach: try to match from all possible text positions.

Helpful terms: text match position and pattern scan index.



Quiz
What are the values of *n,m* in the example? What's the last possible text match position in this case? What was it in example instance?

Algorithm *BruteForceStringMatching*
Output
Input $T[0..n-1], P[0..m-1]$

```
1: for i ← 0 to n − m do
2:     j ← 0
3:     while j < m and P[j] = T[i + j] do
4:         j ← j + 1
5:     if j = m then return i
6: return -1
```

➪ **Operation**

➪ **Efficiency?**

Brute Force String Matching
Understanding

Quiz
How many match positions were tried in example? How many comps were performed?

➪ **Count tried match positions**

➪ **How many comparisons/try?**

➪ **Worst case? When?**

➪ **When better?**

Brute Force
Sequential Search

➪ **Search key**

➪ **Sentinel [value]**

Approach: attempt all possible key matches.

Quiz
What are the best and worst-case efficiencies?

Algorithm *SequentialSearch2*
Input $A[0..n]$ of n keys stored in $0..n-1$, K
Output Index of 1st key to match K, otherwise -1

Improvement ☞

1: $A[n] \leftarrow K$
2: $i \leftarrow 0$
3: **while** $A[i] \neq K$ **do**
4: $\quad i \leftarrow i + 1$
5: **if** $i < n$ **then return** i
6: **return** -1

K acts as a guard (sentinel) against overrunning the list (keep this trick in mind for later).

➪ **Principle**

➪ **Efficiency?**

Brute Force
Bubble Sort

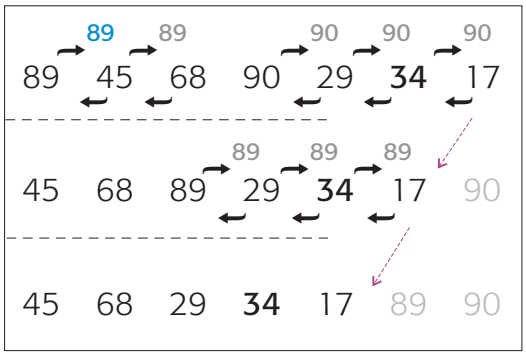
Exercise
Write a pseudocode (design steps) to bubble smaller keys to beginning of list on each round.

Exercise
Complete the example.
Hint: code/log limits and swaps to check answer.

Algorithm *BubbleSort*
Input $A[0..n-1]$
Output

```
1: for i ← 0 to n − 2 do
2:     for j ← 0 to n − 2 − i do
3:         if A[j + 1] < A[j] then
4:             swap A[j], A[j + 1]
```

Check consecutive pairs, correct if not in order; progressively bubble one position back.



• • •

Quiz
Determine loop limits from the small instance without looking at the pseudocode.

Quiz
Compare selection and bubble sort in terms of swaps (hint: how many in the worst case?).

➪ **Efficiency**

➪ **Improvement?**

Usefulness

A brute-force strategy typically goes through steps obvious from definition or clearly needed to solve problem.

Quiz
Did the small improvements in previous examples affect efficiency?

Quiz
What was brute-force about selection sort and bubble sort?

Generally easy, often inefficient, opportunity for small improvement with little effort

- ✎ Low cost (terms of effort to develop)
- ✎ Widely applicable
- ✎ Performance baseline
- ✎ Good enough in some cases (?)