Mansoura University
Faculty of Computers and Information
Department of Computer Science
First Semester: 2020-2021

[MED121] Bioinformatics: String Matching Algorithms

Grade: Third Year (Medical Informatics Program)

Sara El-Metwally, Ph.D.

Faculty of Computers and Information,

Mansoura University,

Egypt.

AGENDA

- Strings Definitions
  - String
  - Substring
  - Suffix
  - Prefix
  - Subsequence
- Exact matching.
- Naïve algorithm for exact matching.
- Lets code!

STRING DEFINITIONS

- A string S is a finite ordered list of characters.
- Characters are drawn from an alphabet Σ
- What is DNA, RNA, and Protein alphabets?

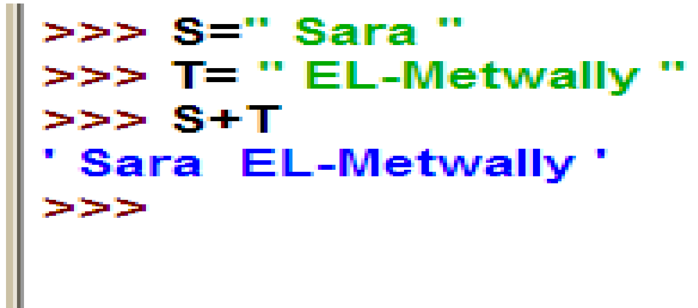
$$\sum proteins = \{A, R, N, D, C, E, Q, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$$

$$\sum DNA = \{A, C, T, G\}$$

$$\sum RNA = \{A, C, U, G\}$$

STRING DEFINITIONS

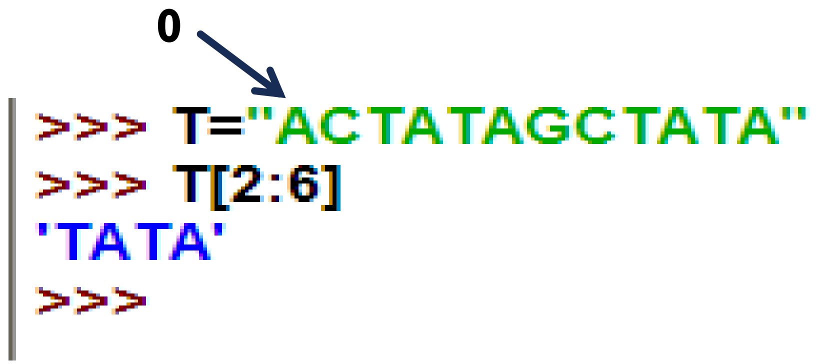
- Length of |S| is a number of characters in S.
- The empty string has a length of what?
- For strings S and T over Σ , their concatenation consists of the characters of S followed by the characters of T, denoted ST.



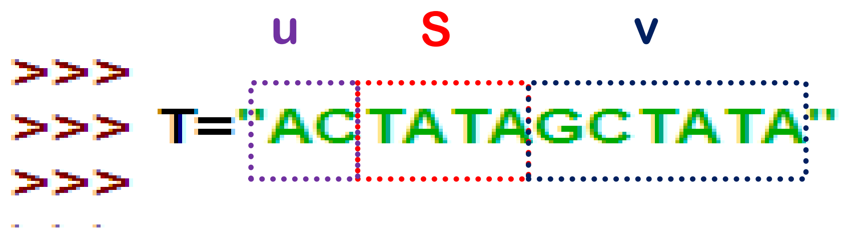
```
>>> S=" Sara "
>>> T= " EL-Metwally "
>>> S+T
' Sara  EL-Metwally '
>>>
```

STRING DEFINITIONS

- S is a substring of T if there exist (possibly empty) strings u and v such that T = uSv.



```
>>> T="ACTATAGCTATA"
>>> T[2:6]
'TATA'
>>>
```



```
>>>
>>> T="ACTATAGCTATA"
>>>
```

STRING DEFINITIONS

- Find all possible substrings of “ATATGC”?

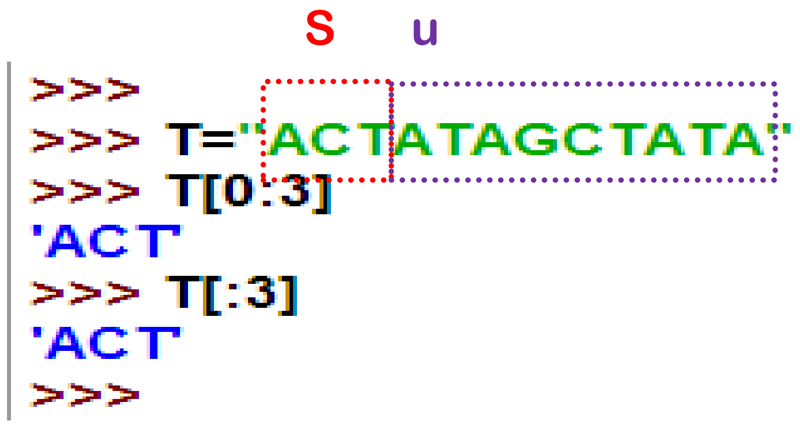
  A, AT, ATA, ATAT, ATATG, ATATGC

  T, TA, TAT,…

  A, AT, ATG,…

STRING DEFINITIONS

- S is a prefix of T if there exists a string u such that T = Su.



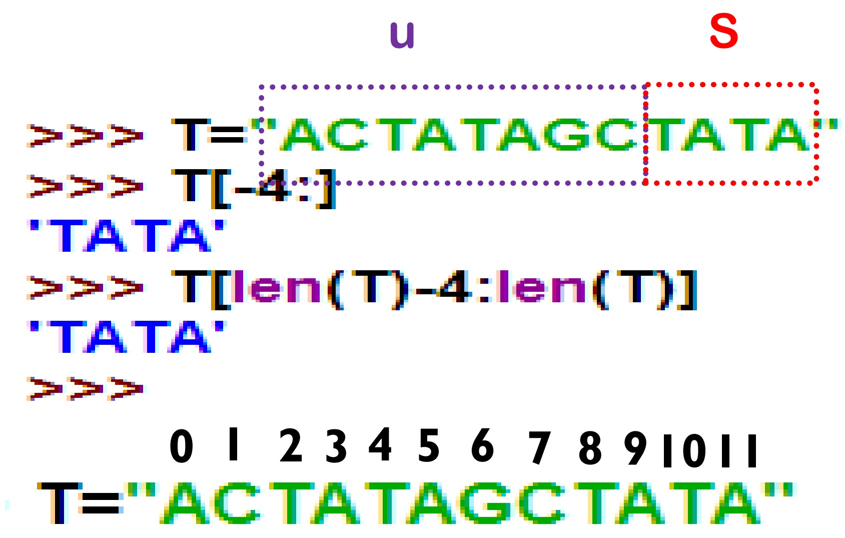
```
>>>
>>> T="ACTATAGCTATA"
>>> T[0:3]
'ACT'
>>> T[:3]
'ACT'
>>>
```

✓Find all possible prefixes of “ATATGC”?

A, AT, ATA, ATAT, ATATG, ATATGC

STRING DEFINITIONS

- S is a suffix of T if there exists a string u such that T = uS.



```
>>> T="ACTATAGCTATA"
>>> T[-4:]
'TATA'
>>> T[len(T)-4:len(T)]
'TATA'
>>>
```

```
    0 1 2 3 4 5 6 7 8 9 10 11
T="A C T A T A G C T A T  A"
```

✓Find all possible suffixes of "ATATGC"?

C, GC, TGC, ATGC, TATGC, ATATGC

STRING DEFINITIONS

- **Subsequence** is similar to substring except the characters need not be consecutive.

String = " ACCCTTTTATTGT "

Substring = "CTTTT "

Subsequence = " ACATT "

EXACT MATCHING

- **Inputs:** Pattern P, Text T.
- **Output:** find all occurrences of P in T.
- **Goal:** Looking for places where a pattern P occurs as a substring of a text T. Each such place is an occurrence or match.

CTTTTGTATATTTATAGCTTTTATAGCCC , Text T

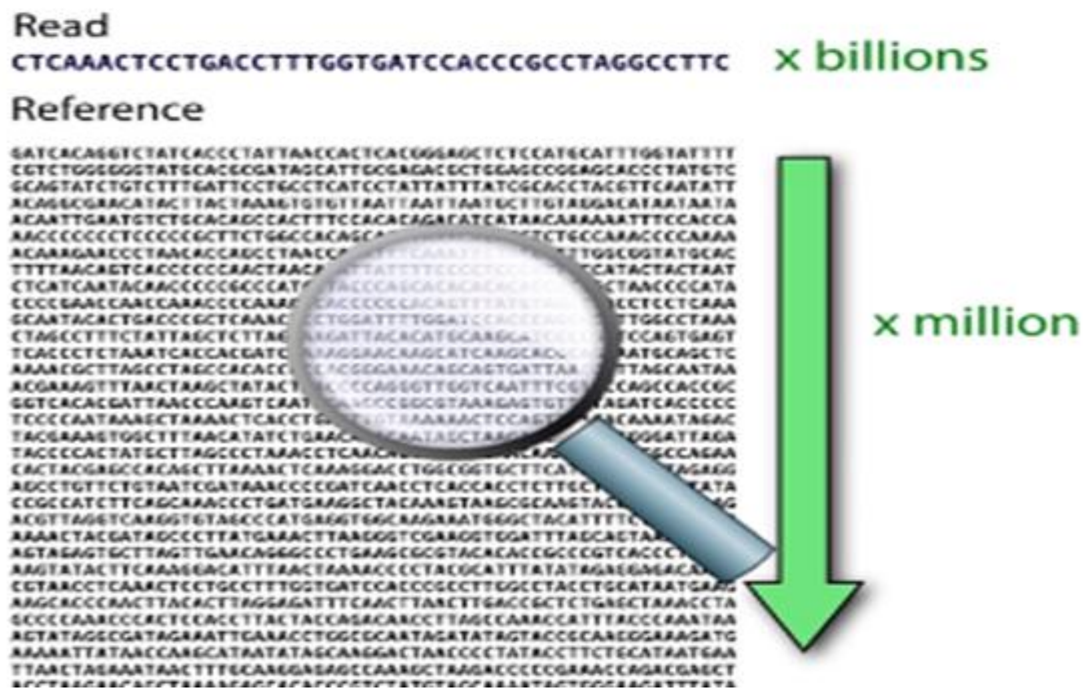
GTATAT, Pattern P

EXACT MATCHING

Read
CTCAAACTCCTGACCTTTGGTGATCCACCCGCCTAGGCCTTC
Reference

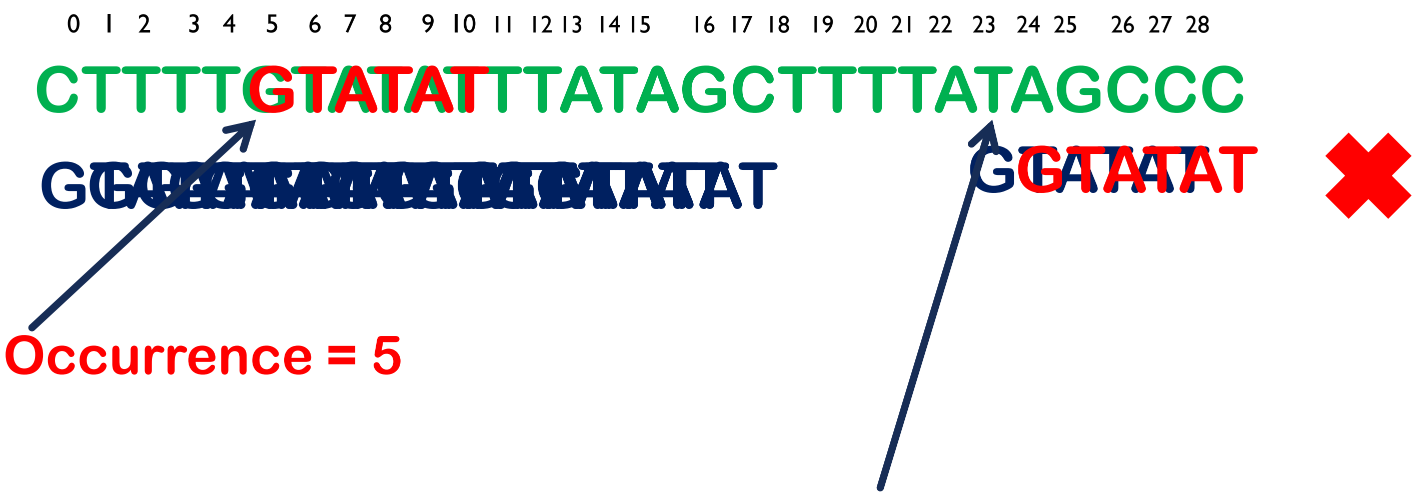
Slide Credit: Ben Langmead course of Algorithms for DNA Sequencing

EXACT MATCHING



Slide Credit: Ben Langmead course of Algorithms for DNA Sequencing

EXACT MATCHING



Occurrence = 5

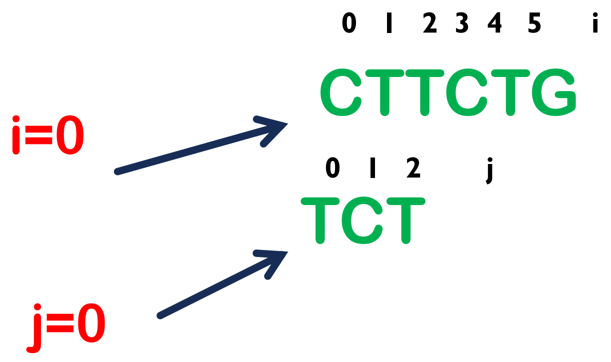
Last place to compare = 23

Len(T) – Len(P)

29 -6=23

EXACT MATCHING (NAÏVE ALGORITHM)

- Simple algorithm: Try all possible alignments.
- For each, check whether it's an occurrence or not.



match = true

T[i+j]=T[0]=C

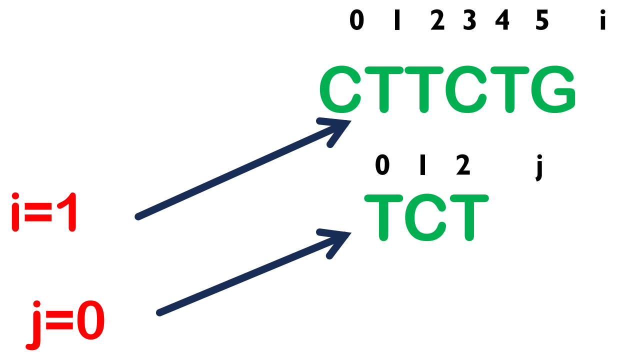
P[j]=P[0]=T

match = false

Break j loop

EXACT MATCHING (NAÏVE ALGORITHM)

- **Simple algorithm:** Try all possible alignments.
- For each, check whether it's an occurrence or not.



match = true

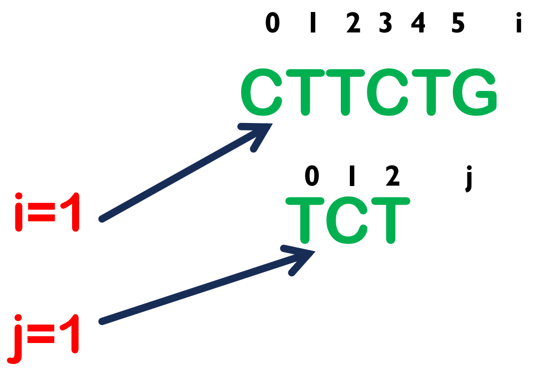
T[i+j]=T[1]=T

P[j]=P[0]=T

Continue j loop

EXACT MATCHING (NAÏVE ALGORITHM)

- Simple algorithm: Try all possible alignments.
- For each, check whether it's an occurrence or not.



match = true

T[i+j]=T[2]=T

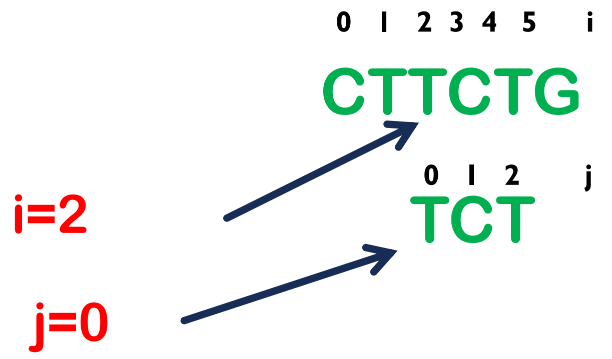
P[j]=P[1]=C

match = false

Break j loop

EXACT MATCHING (NAÏVE ALGORITHM)

- Simple algorithm: Try all possible alignments.
- For each, check whether it's an occurrence or not.



match = true

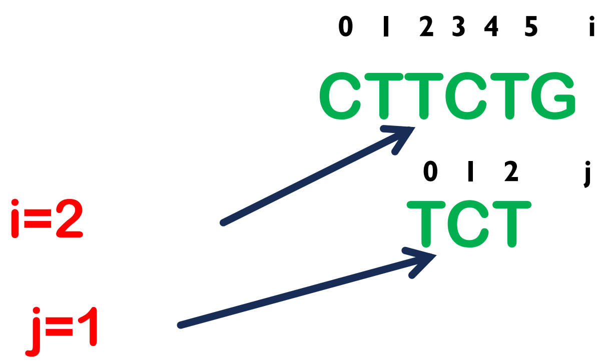
T[i+j]=T[2]=T

P[j]=P[0]=T

Continue j loop

EXACT MATCHING (NAÏVE ALGORITHM)

- Simple algorithm: Try all possible alignments.
- For each, check whether it's an occurrence or not.



match = true

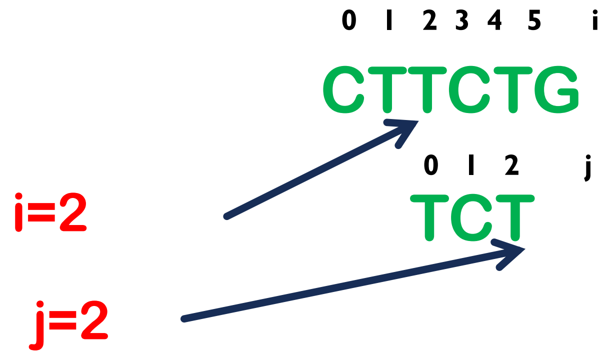
T[i+j]=T[3]=C

P[j]=P[1]=C

Continue j loop

EXACT MATCHING (NAÏVE ALGORITHM)

- **Simple algorithm:** Try all possible alignments.
- For each, check whether it's an occurrence or not.



match = true

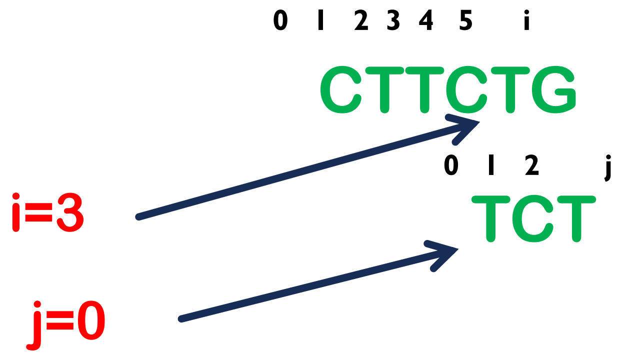
T[i+j]=T[4]=T

P[j]=P[2]=T

# end of pattern
# match = true
# occurrence = i = 2

EXACT MATCHING (NAÏVE ALGORITHM)

- Simple algorithm: Try all possible alignments.
- For each, check whether it's an occurrence or not.



match = true

T[i+j]=T[3]=C

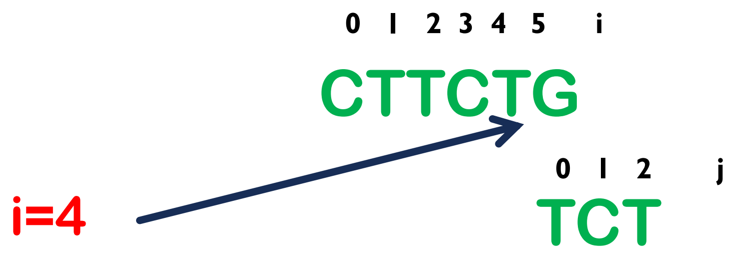
P[j]=P[0]=T

✗ match = false

Break j loop

EXACT MATCHING (NAÏVE ALGORITHM)

- Simple algorithm: Try all possible alignments.
- For each, check whether it's an occurrence or not.



i = len(T) – len (P) +1
i= 6-3+1= 4

Stop i loop !

EXACT MATCHING (NAÏVE ALGORITHM)

```
def naive(p, t):
    assert len(p) <= len(t)
    occurrences = []
    for i in range(0, len(t)-len(p)+1):
        match = True
        for j in range(0, len(p)):
            if t[i+j] != p[j]:
                match = False
                break
        if match:
            occurrences.append(i)
    return occurrences

if __name__ == '__main__':
 #t="CTTCTGTCTGGGTCT"
 t="CTTCTG"
 p="TCT"
 occurrences = []
 occurrences = naive(p, t)
 print(occurrences)
 i=occurrences[0]
 j=i+len(p)
 print(t[i:j])
 '''
 for i in range(0, len(occurrences)):
    x=occurrences[i]
    y=x+len(p)
    print(x)
    print(t[x:y])'''
```

NAÏVE ALGORITHM (TIME COMPLEXITY)

- Algorithm running time = # of operations to be executed.
- The greater the number of operations, the longer the running time of an algorithm.
- We usually want to know how many operations an algorithm will execute in proportion to the size of its input .
- Big O specifically describes the worst-case scenario.

NAÏVE ALGORITHM (TIME COMPLEXITY) (HTTP://BIGOCHEATSHEET.COM/)

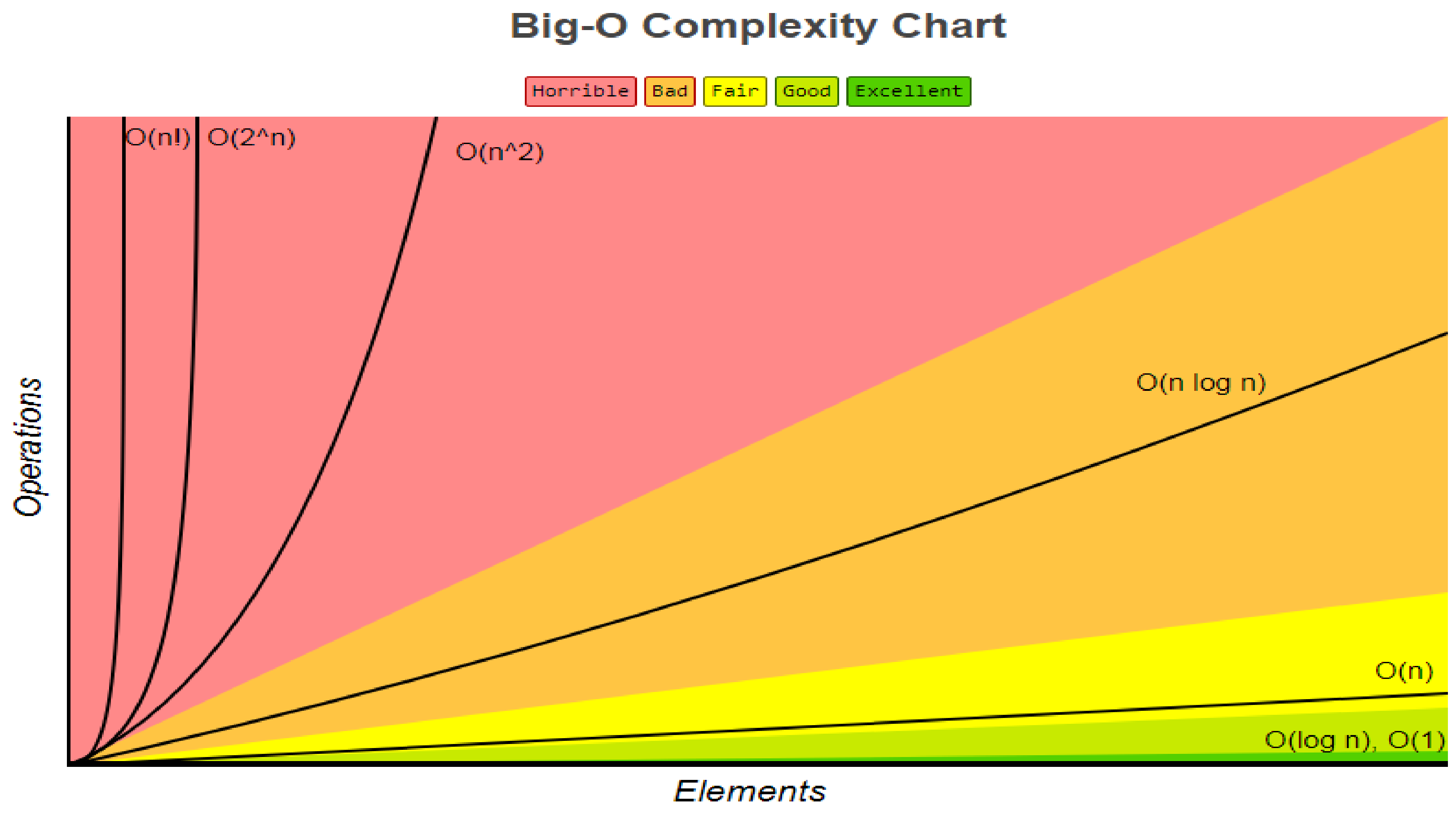
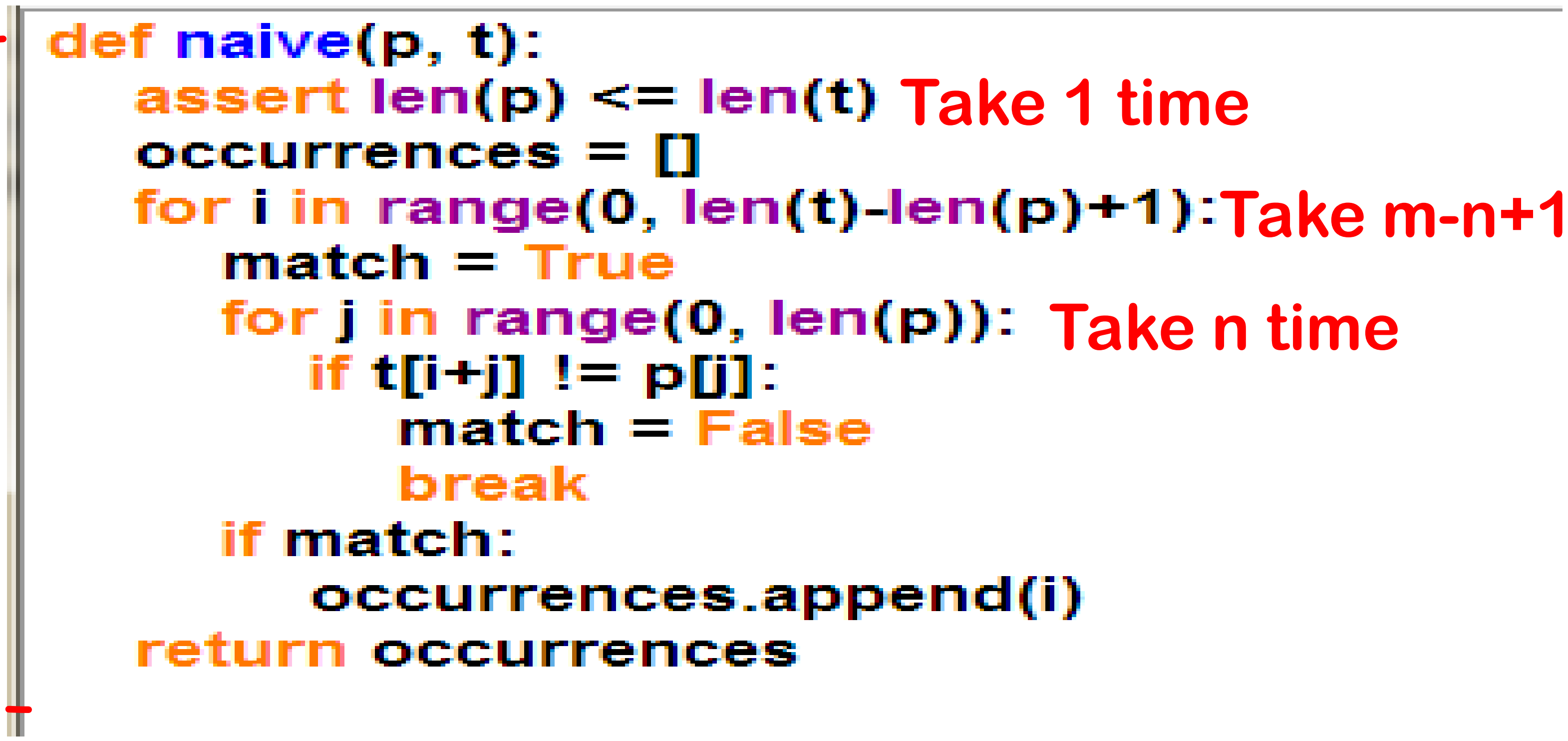


Image Credit: https://www.bigocheatsheet.com/

NAÏVE ALGORITHM

- let m = | T |, and Let n = | P | , and assume n ≤ m



```
def naive(p, t):
    assert len(p) <= len(t)
    occurrences = []
    for i in range(0, len(t)-len(p)+1):
        match = True
        for j in range(0, len(p)):
            if t[i+j] != p[j]:
                match = False
                break
        if match:
            occurrences.append(i)
    return occurrences
```

Take 1 time

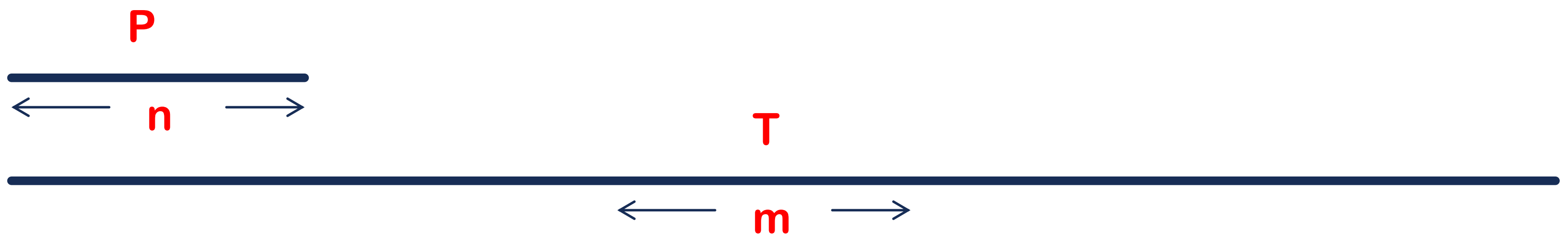
Take m-n+1 time

Take n time

Take n(m-n+1)

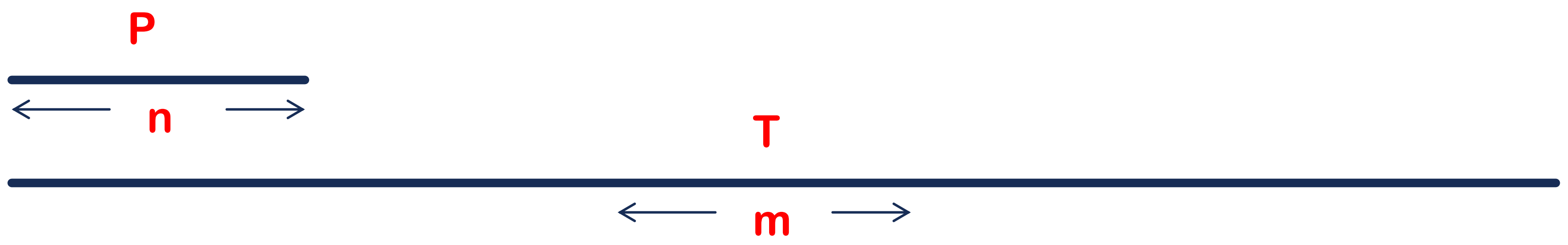
O(nm)

Q

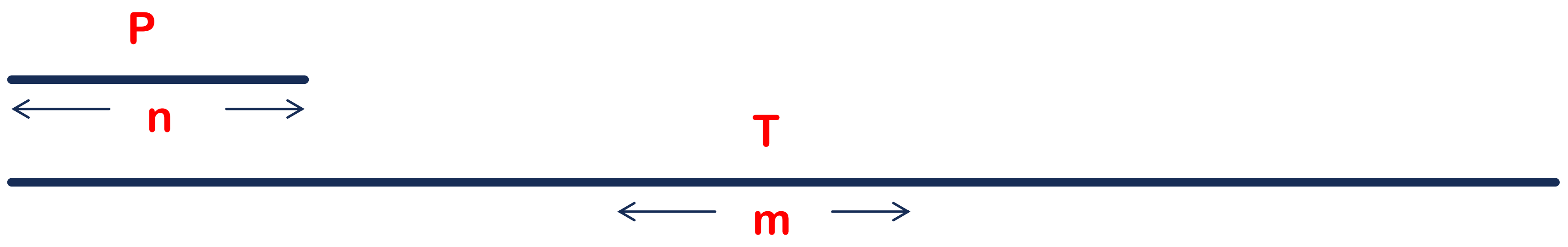


- How many alignments are possible between text and pattern ?
- What is the greatest number of characters comparisons possible ? Give me example of pattern and text.
- What is the least number of characters comparisons possible? Give me example of pattern and text.

ANSWERS.



Q



- How many alignments are possible between text and pattern ?

  m-n+1

- What is the greatest number of characters comparisons possible ? Give me example of pattern and text.

  n(m-n+1)

- What is the least number of characters comparisons possible? Give me example of pattern and text.

  m-n+1

Thank you!