Printing all substrings

" "
a
b
c
d
ab
ac
ad
.
.
.

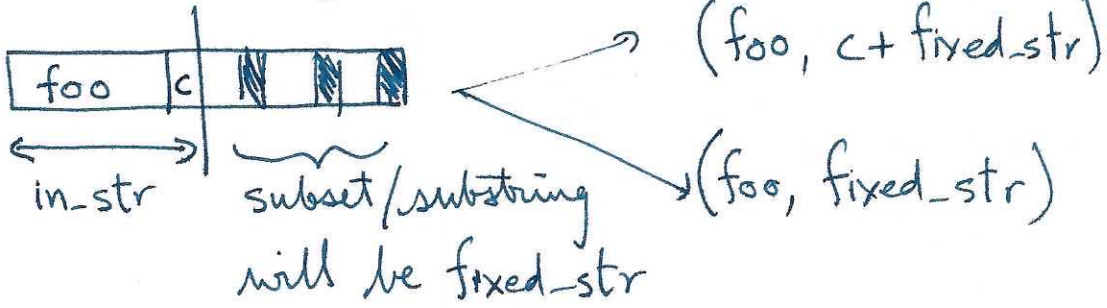
"abcd"
↔
4

$16 = 2^4$ different substrings

→ How to generate all substrings ←
For each character, it is either present or not

void printAllSub(string s)

void printAllSub(string in_str, string fixed_str)



Base case: when in_str is empty,
print fixed_str

Recursion: deciding whether to keep the
last character in in_str
(fixed_str ~~keeps~~ tracks all characters
that are kept)

```
void printAllSub (string in_str, string fixed_str)
{
    if (length(in_str) == 0)
    {
        print fixed_str;
        return;
    }

    char c = last character of in_str;
    delete last character from in_str;
    printAllSub(in_str, c + fixed_str);
    printAllSub(in_str, fixed_str);
}

void printAllSub(string input)
{
    printAllSub(input, "");
}
```

base case

appending to beginning

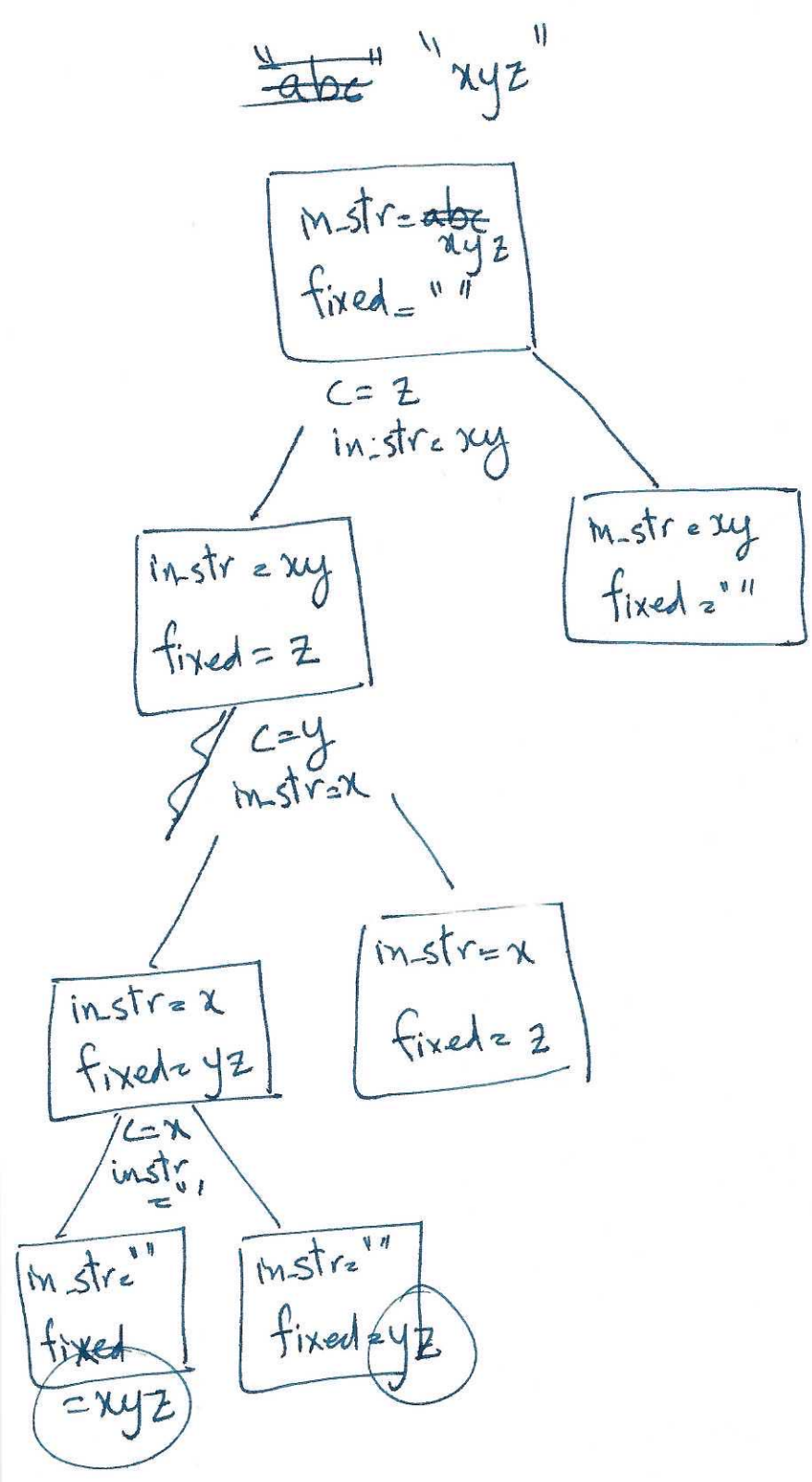
abcd

$2^4 = 16$ substrings

Q. Which is the FIRST substring printed?

(A) input
(B) Empty String
(C) Something else

How many recursive calls if input has n characters? Essentially $2^{n+1} - 1$


"abc" "xyz"
in_str="abc" xyz
fixed=""
c=z
in_str=xy
in_str=xy
fixed=z
in_str=xy
fixed=""
c=y
in_str=x
in_str=x
fixed=yz
in_str=x
fixed=z
c=x
in_str=""
in_str=""
fixed
=xyz
in_str=""
fixed=yz

$f, g : \mathbb{N} \to \mathbb{N}$

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$, we say $f(n) = O(g(n))$ (→ big-Oh of $g(n)$)

"$f \leq g$"

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$, we say $f(n) = \Omega(g(n))$

"$f \geq g$"

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \neq 0$, $f(n) = \Theta(g(n))$ "$f = g$"

$f(n) = T_{\text{selection sort}}(n)$ $\quad g(n) = n$ $\quad g(n) = n^2$

Q. Can $f = O(g)$ and $f = \Omega(g)$?

~~(R)~~ Yes (G) No $\quad f = g \quad f = \Theta(g)$

Q. Is $n = O(2n+3)$? ~~(R)~~ Yes (G) No

$$\lim_{n\to\infty} \frac{n}{2n+3} = 1/2 < \infty$$

Q. Is $n = O\left(\frac{n}{2}+3\right)$? ~~(R)~~ Yes (G) No

$$\lim_{n\to\infty} \frac{n}{n/2+3} = 2 < \infty$$

We compare $T_A(n)$ vs $T_B(n)$ as $n \to \infty$,
asymptotically (in the limit)

We compare the rates of growth.

$$\lim_{n\to\infty} \frac{T_A(n)}{T_B(n)}$$ (Assume limit exists)

0

"$T_A(n) < T_B(n)$"

"A is faster"

$c \neq 0$

(some non-zero constant)

"$T_A(n) = T_B(n)$"

"A is as fast as B and vice versa"

∞

"$T_A > T_B$"

Given an input x of size n, count the number of "elementary" operations that an algorithm A performs on x.

(Array → x; length → n)

A = selection sort — x = array — n = length

A = searching in a list — x = list, and element to be found — n = length

Running time of A on x = $RT_A(x)$

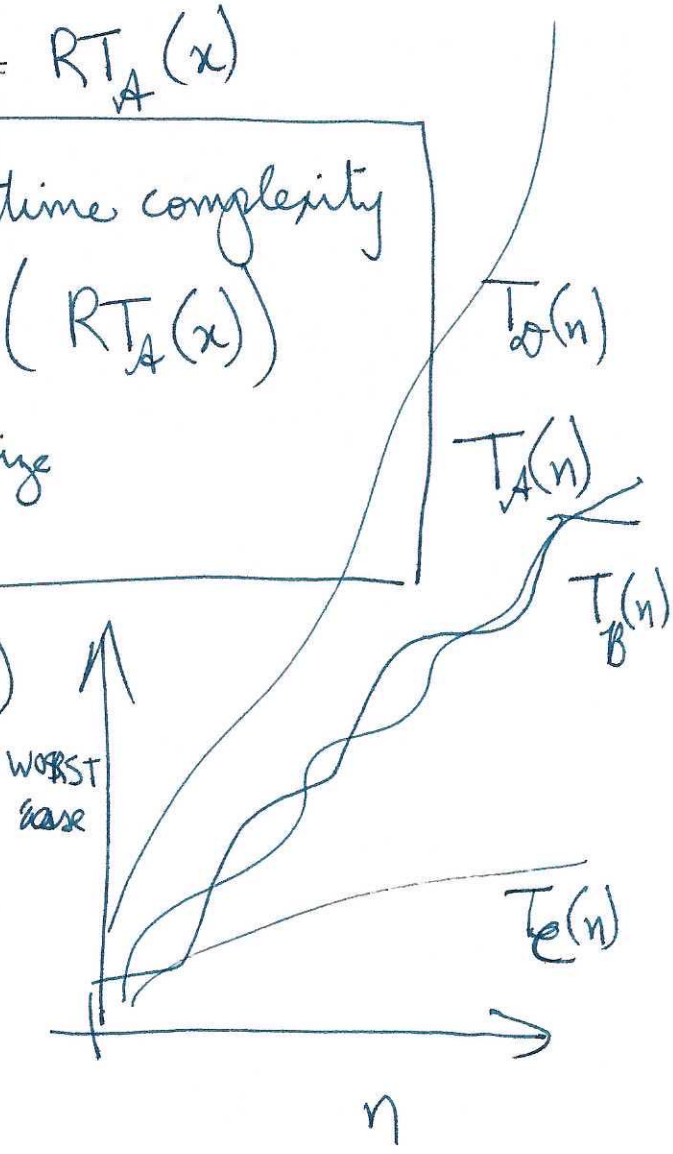
WORST case running time / time complexity

$$T_A(n) = \max_{x:\ x \text{ has size } n} \left(RT_A(x) \right)$$

(n ← input size)

$T_A(n)$ vs $T_B(n)$

different algorithms for same task



Asymptotic Analysis of Algorithms

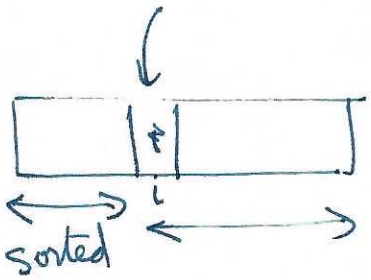
Sorting an array of ints

A is an array of length n

Find minimum in A
Swap with A[0]
Find min in A[1..n-1]
Swap with A[1]
⋮

} Selection sort

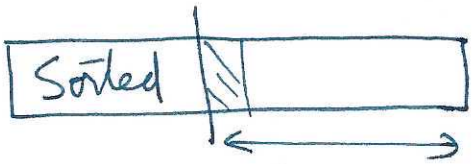
```
for i=0 to n-1
{
    min = A[i]
    minindex = i
    for j=i to n-1
    {
      if (A[j] < min)
      {
        min = A[j]; minindex = j;
      }
    }
    swap A[i] with A[minindex]
}
```



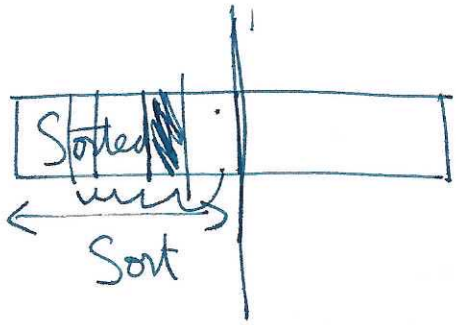
} initialization

} Find min in A[i..n-1]

Selection



Insertion



Insertion Sort

```
for i=0 to n-1
{
    j=i
    while (j>0 && A[j-1] > A[j])
    {
        swap A[j-1] and A[j]
        j--;
    }
}
```

How to compare/determine the "efficiency" of different algorithms for the same problem?

How to determine the best data structure for a task? → most "efficient"

Q. Is $5n^3+10n = O(n^2)$? (R) Yes ~~(G)~~ No

$$\lim_{n\to\infty} \frac{5n^3+10n}{n^2} = \infty$$

Q. Is $2n+3 = O(n^2)$? ~~(R)~~ Yes (G) No

$$\lim_{n\to\infty} \frac{2n+3}{n^2} = 0 < \infty$$ But

$2n+3 \neq \Theta(n^2)$

Q. Is $\log_2 n = O(n)$? ~~(R)~~ Yes (G) No

$$\lim_{n\to\infty} \frac{\log_2 n}{n} = 0$$ (L'Hopital's rule)

Q. Is $\log_2 n = O(\sqrt{n})$? ~~(R)~~ Yes (G) No

$T(n) = 8n^2 + 5n + 100$ ← Lower order term

Leading constant

$T(n) = O(n^2)$ $T(n) = \Theta(n^2)$