Data Structures and Algorithms

(CS210A)

Lecture 21

- Analyzing average running time of Quick Sort

Overview of this lecture

Main Objective:

- Analyzing average time complexity of **QuickSort** using **recurrence**.
  - Using mathematical induction.
  - Solving the recurrence exactly.
- The outcome of this analysis will be quite surprising!

Extra benefits:

- You will learn a standard way of using mathematical induction to bound time complexity of an algorithm. You must try to internalize it.

QuickSort

Pseudocode for QuickSort(S)

```
QuickSort(S)
{       If (|S|>1)
                Pick and remove an element x from S;
                (S_<x, S_>x) ← Partition(S,x);
                return( Concatenate(QuickSort(S_<x), x, QuickSort(S_>x))
}
```

Pseudocode for QuickSort(S)

When the input S is stored in an array

```
QuickSort(A,l, r)
{       If (l < r)
                i← Partition(A,l,r);
                QuickSort(A,l, i − 1);
                QuickSort(A,i + 1, r)
}
```

Partition :

x← $A[l]$ as a pivot element,

permutes the subarray $A[l \dots r]$ such that

elements preceding x are smaller than x,

$A[i]$= x,

and elements succeeding x are greater than x.

Analyzing average time complexity of QuickSort

Part 1

Deriving the recurrence

Analyzing average time complexity of QuickSort

Assumption (just for a neat analysis):

- All elements are **distinct**.
- Each recursive call selects the **first element** of the subarray as the pivot element.

Analyzing average time complexity of QuickSort

A useful Fact: **Quick sort** is a comparison based algorithm.

0	1	2	3	4	5	6	7	8
6	**11**	**42**	**37**	**24**	**5**	**16**	**27**	**2**
e_3	e_4	e_9	e_8	e_6	e_2	e_5	e_7	e_1

0	1	2	3	4	5	6	7	8
15	**20**	**49**	**41**	**29**	**4**	**23**	**36**	**3**
e_3	e_4	e_9	e_8	e_6	e_2	e_5	e_7	e_1

Let e_i : ith **smallest** element of $\boldsymbol{A}$.

Observation: The execution of **Quick sort** depends upon the permutation of e_i's and **not** on the values taken by e_i's.

Analyzing average time complexity of QuickSort

$\boldsymbol{T}(\boldsymbol{n})$: Average running time for Quick sort on input of size $\boldsymbol{n}$.

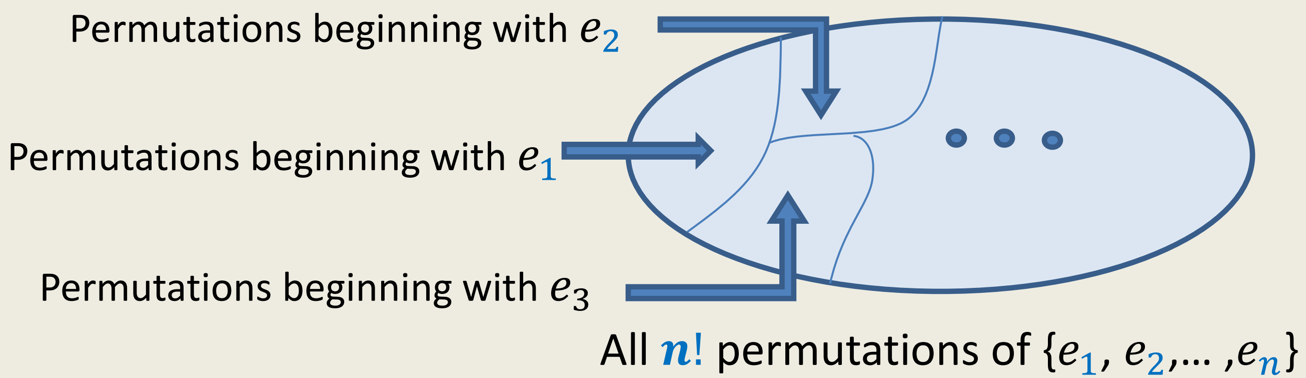
(average over all possible permutations of $\{e_1, e_2, \ldots, e_n\}$)

Hence, $\boldsymbol{T}(\boldsymbol{n}) = \frac{1}{\boldsymbol{n}!} \sum_{\pi} \boldsymbol{Q}(\pi)$,

where $\boldsymbol{Q}(\pi)$ is the time complexity (or no. of comparisons) when the input is permutation π.

Calculating $\boldsymbol{T}(\boldsymbol{n})$ from definition/scratch is impractical, if not impossible.

Analyzing average time complexity of QuickSort



Let **P**(i) be the set of all those permutations of $\{e_1, e_2, \ldots, e_n\}$ that begin with e_i.

Question: What fraction of all permutations constitutes **P**(i) ?

Answer: $\frac{1}{n}$

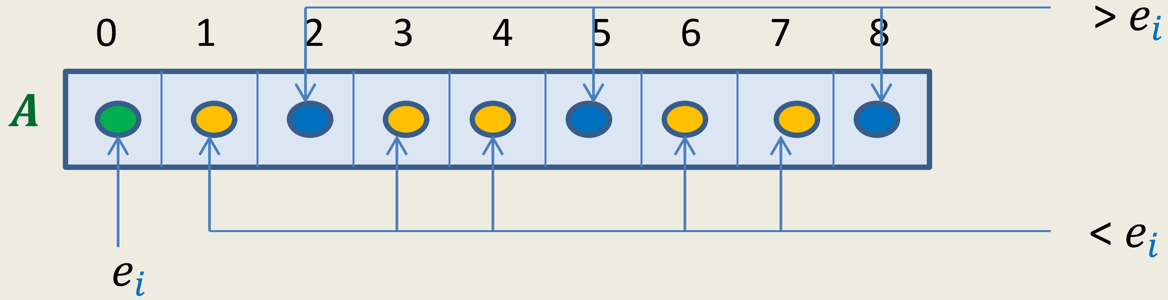
Let $G(n, i)$ be the average running time of **QuickSort** over **P**(i).

Question: What is the relation between $T(n)$ and $G(n, i)$'s ?

Answer: $T(n) = \frac{1}{n}\sum_{i=1}^{n} G(n, i)$

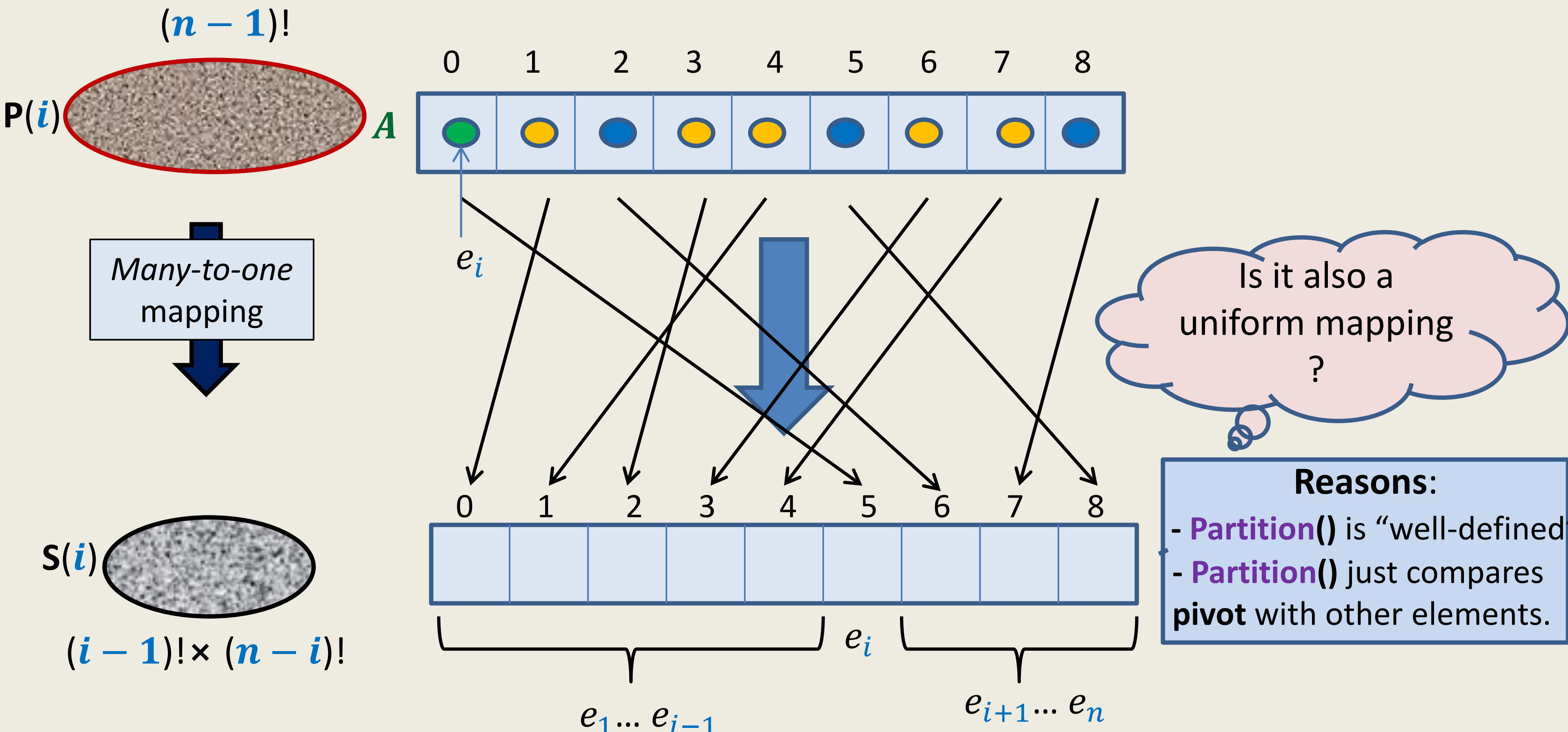
Observation: We now need to derive an expression for $G(n, i)$.For this purpose, we need to have a closer look at the execution of **QuickSort** over **P**(i).

Quick Sort on a permutation from P(i).



What happens during **Partition**(A,0,8)

Quick Sort on a permutation from **P**(i).



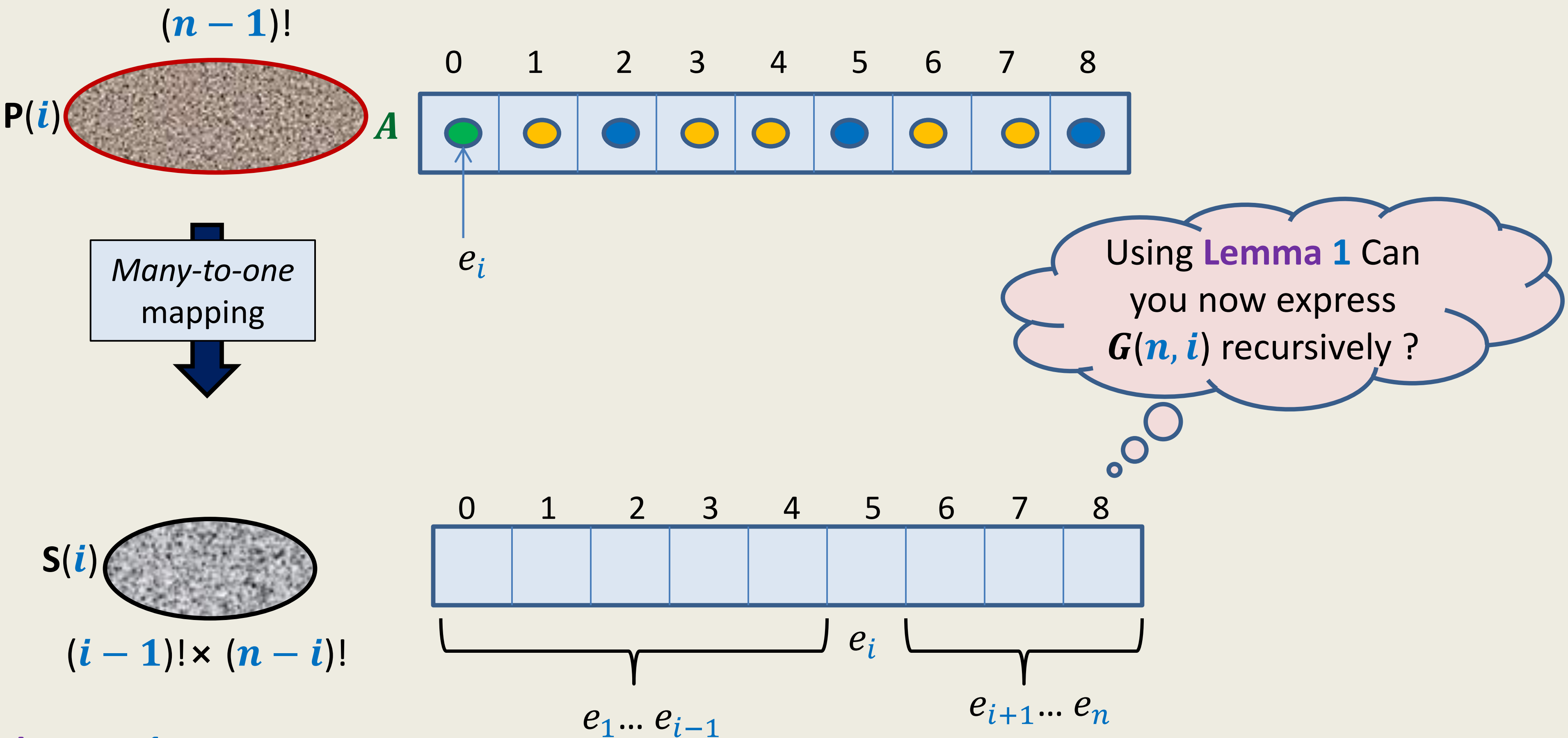
Lemma 1:

S(i): Permutations resulting from **Partition**().

There are <u>exactly</u> $\binom{n-1}{i-1}$ permutations from P(i) that get mapped to one permutation in **S**(i).

Homework: Try your best to prove **Lemma 1**. If you have still any doubt, please meet me.

Quick Sort on a permutation from P(i).



Lemma 1:

There are <u>exactly</u> $\binom{n-1}{i-1}$ permutations from **P**(i) that get mapped to one permutation in **S**(i).

Analyzing average time complexity of QuickSort

$G(n, i)$ =

$$T(i-1) + T(n-i) \;+\; \mathrm{d}n \qquad \text{----1}$$

We showed previously that :

$$T(n) = \frac{1}{n}\sum_{i=1}^{n} G(n, i) \qquad \text{----2}$$

Question: Can you express $T(n)$ recursively using **1** and **2**?

$$T(n) = \frac{1}{n}\sum_{i=1}^{n}\big(T(i-1) + T(n-i)\big) + \mathrm{d}n$$

$$T(1) = \mathrm{c}$$

Analyzing average time complexity of QuickSort

Part 2

Solving the recurrence through mathematical induction

$T(1) \quad = c$

$T(n) \quad = \frac{1}{n}\sum_{i=1}^{n}(T(i-1)+T(n-i)) \;+\; dn$

$\quad\quad = \frac{2}{n}\sum_{i=1}^{n-1} T(i) \;+\; dn$

Assertion A(m): $T(m) \le am \log m + b$ for all $m \ge 1$

Base case A(0) : Holds for $b \ge c$

Induction step: Assuming **A(m)** holds for all $m < n$, we have to prove **A(n).**

$T(n) \le \frac{2}{n}\sum_{i=1}^{n-1}(ai \log i + b) \;+\; dn$

$\le \frac{2}{n}(\sum_{i=1}^{n-1} ai \log i\,) \;+\; 2b + dn$

$= \frac{2}{n}(\sum_{i=1}^{n/2} ai \log i \;+ \sum_{i=\frac{n}{2}+1}^{n-1} ai \log i\,) + 2b + dn$

$\le \frac{2}{n}(\sum_{i=1}^{n/2} ai \log n/2 + \sum_{i=\frac{n}{2}+1}^{n-1} ai \log n\,) + 2b + dn$

$= \frac{2}{n}(\sum_{i=1}^{n-1} ai \log n \;-\; \sum_{i=1}^{n/2} ai) \;+\; 2b + dn$

$= \frac{2}{n}\,(\frac{n(n-1)}{2}\, a \log n \;-\; \frac{\frac{n}{2}(\frac{n}{2}+1)}{2}\, a) + \; 2b + dn$

$\le a(n-1)\log n \;-\frac{n}{4}\, a + 2b + dn$

$= an\log n + b - \frac{n}{4}\, a \;+ b + dn$

$\le an\log n + b \qquad$ **for** $a > 4(b + d)$

Analyzing average time complexity of QuickSort

Part 3

Solving the recurrence exactly

Some elementary tools

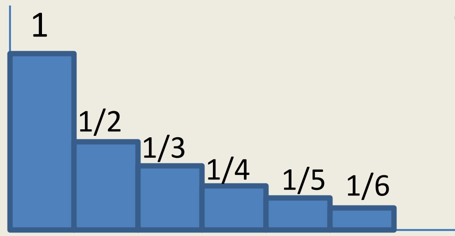
$$\mathbf{H}(n) = \sum_{i=1}^{n} \frac{1}{i}$$

Question: How to approximate $\mathbf{H}(n)$ **?**

Answer: $\mathbf{H}(n) \rightarrow \log_e n + \gamma$, as n increases

where γ is **Euler's constant ~0.58**

Hint: →



Look at this figure, and relate it to the curve for function f(x)= 1/x and its integration…

We shall calculate average number of comparisons during QuickSort using:

- our knowledge of solving recurrences by substitution
- our knowledge of solving recurrence by unfolding
- our knowledge of simplifying a partial fraction (from JEE days)

Students should try to internalize the way the above tools are used.

$T(n)$: average number of <u>comparisons</u> during **QuickSort** on n elements.

$T(1) = 0$, $T(0) = 0$,

$T(n) = \frac{1}{n}\sum_{i=1}^{n}(T(i-1) + T(n-i)) + n - 1$

$= \frac{2}{n}\sum_{i=1}^{n}(T(i-1)) + n - 1$

➔ $nT(n) = 2\sum_{i=1}^{n}(T(i-1)) + n(n-1)$ -----**1**

Question: How will this equation appear for $n-1$?

$(n-1)T(n-1) = 2\sum_{i=1}^{n-1}(T(i-1)) + (n-1)(n-2)$ -----2

Subtracting 2 from 1, we get

$nT(n) - (n-1)T(n-1) = 2\,T(n-1) + 2(n-1)$

➔ $nT(n) - (n+1)T(n-1) = 2(n-1)$

Question: How to solve/simplify it further ?

$$\frac{T(n)}{n+1} - \frac{T(n-1)}{n} = \frac{2(n-1)}{n(n+1)}$$

$$\frac{T(n)}{n+1} - \frac{T(n-1)}{n} = \frac{2(n-1)}{n(n+1)}$$

➔ $g(n) - g(n-1) = \frac{2(n-1)}{n(n+1)}$, where $g(m) = \frac{T(m)}{m+1}$

Question: How to simplify RHS ?

$$\frac{2(n-1)}{n(n+1)} = \frac{2(n+1)-4}{n(n+1)} =$$

$$= \frac{2}{n} - \frac{4}{n(n+1)}$$

$$= \frac{2}{n} - \frac{4}{n} + \frac{4}{n+1}$$

$$= \frac{4}{n+1} - \frac{2}{n}$$

➔ $g(n) - g(n-1) = \frac{4}{n+1} - \frac{2}{n}$

$$g(n) - g(n-1) = \frac{4}{n+1} - \frac{2}{n}$$

Question: How to calculate $g(n)$?

$$g(n-1) - g(n-2) = \frac{4}{n} - \frac{2}{n-1}$$

$$g(n-2) - g(n-3) = \frac{4}{n-1} - \frac{2}{n-2}$$

$$\ldots \quad = \quad \ldots$$

$$g(2) - g(1) = \frac{4}{3} - \frac{2}{2}$$

$$g(1) - g(0) = \frac{4}{2} - \frac{2}{1}$$

Hence $g(n) = \frac{4}{n+1} + (2\sum_{j=2}^{n} \frac{1}{j}) - 2 \quad = \frac{4}{n+1} + (2\sum_{j=1}^{n} \frac{1}{j}) - 4$

$$= \frac{4}{n+1} + 2\mathrm{H}(n) - 4$$

➔ $T(n) = (n+1)\left(\frac{4}{n+1} + 2\mathrm{H}(n) - 4\right)$

$$= 2(n+1)\mathrm{H}(n) - 4n$$

$$\begin{aligned} T(n) &= 2(n+1)\mathrm{H}(n) - 4n \\ &= 2(n+1)\log_e n + 1.16\,(n+1) - 4n \\ &= 2n\log_e n - 2.84\,n + \mathrm{O}(1) \\ &= 2n\log_e n \end{aligned}$$

Theorem: The average number of comparisons during **QuickSort** on n elements approaches $2n\log_e n - 2.84\,n$.

$$= 1.39\,n\log_2 n - \mathrm{O}(n)$$

The best case number of comparisons during **QuickSort** on n elements = $n\log_2 n$

The worst case no. of comparisons during **QuickSort** on n elements = $n(n-1)$

Quick sort versus Merge Sort

No. of Comparisons	Merge Sort	Quick Sort
Average case	$n \log_2 n$	$1.39\, n \log_2 n$
Best case	$n \log_2 n$	$n \log_2 n$
Worst case	$n \log_2 n$	$n(n-1)$

After seeing this table, no one would prefer **Quick sort** to **Merge sort**

But **Quick sort** is still the most preferred algorithm in practice. Why ?

You will find the answer **yourself** in the next **programming assignment** ☺