Quick Sort and Full History Recurrences

Subhabrata Samajder

IIIT, Delhi

Summer Semester,
9^{th} May, 2022

Quicksort

Motivation

Recall:

- Mergesort needs extra storage.
- It is not possible to predict where each element will end up in the final order.

Motivation

Recall:

- Mergesort needs extra storage.
- It is not possible to predict where each element will end up in the final order.

Question: Can we somehow perform a different divide and conquer so that the position of the elements can be determined?

Motivation

Recall:

- Mergesort needs extra storage.
- It is not possible to predict where each element will end up in the final order.

Question: Can we somehow perform a different divide and conquer so that the position of the elements can be determined?

Basic Idea of Quicksort:

- Spend most of the effort in the divide step and
- very little in the conquer step!

The Divide and Combine Step

- **The Divide Step:**
  - Suppose that we know a number x such that *one-half* of the elements are $> x$ and the *other-half* of the elements are $\leq x$.
  - Compare all elements to x.
  - Partition the sequence into two parts according to the answer.
  - This partition requires $n - 1$ comparisons.
  - One part can occupy the first half of the array and the other the second half.
  - $\therefore$ can be done *in-place*.

The Divide and Combine Step

- **The Divide Step:**
  - Suppose that we know a number x such that *one-half* of the elements are $> x$ and the *other-half* of the elements are $\leq x$.
  - Compare all elements to x.
  - Partition the sequence into two parts according to the answer.
  - This partition requires $n - 1$ comparisons.
  - One part can occupy the first half of the array and the other the second half.
  - $\therefore$ can be done *in-place*.

- Then sort each subsequence recursively.

The Divide and Combine Step

- **The Divide Step:**
  - Suppose that we know a number x such that *one-half* of the elements are $> x$ and the *other-half* of the elements are $\leq x$.
  - Compare all elements to x.
  - Partition the sequence into two parts according to the answer.
  - This partition requires $n - 1$ comparisons.
  - One part can occupy the first half of the array and the other the second half.
  - $\therefore$ can be done *in-place*.

- Then sort each subsequence recursively.

- **The Combine Step:** Trivial!
  - The two parts already occupy the correct positions in the array.
  - Therefore, no additional space is required.

How To Find x?

- Till now, it was assumed that the value of x is known.
- However, x is usually unknown.

How To Find x?

- Till now, it was assumed that the value of x is known.
- However, x is usually unknown.
- **Note:** It is easy to see, that the same algorithm will work no matter which number is used for the *partition*.
- Call the number x as the *pivot*.

How To Find x?

- Till now, it was assumed that the value of x is known.
- However, x is usually unknown.
- **Note:** It is easy to see, that the same algorithm will work no matter which number is used for the *partition*.
- Call the number x as the *pivot*.
- Our purpose is to partition the array into two parts,
  - one with numbers $>$ than the pivot and
  - the other with numbers $\leq$ the pivot.
- This is achieved via the partitioning algorithm.

The Partition Algorithm

- Use two pointers to the array, L and R.
- Initially,
  - L points to the left side of the array and
  - R points to the right side of the array.
- The pointers "move" in opposite directions toward each other.
- Swap(x_L, x_R): If $x_L > pivot$ and $x_R \leq pivot$.

Quicksort: An Example

The Partitioning Phase:

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14

The Partitioning Phase:

pivot

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14

L *R*

Quicksort: An Example

The Partitioning Phase:

pivot															
6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
	L														*R*

Quicksort: An Example

The Partitioning Phase:

pivot

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14

L *R*

The Partitioning Phase:

pivot ↓

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
		↑ L (*pause*)													↑ R

Quicksort: An Example

The Partitioning Phase:

pivot

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14

L (*pause*) *R*

Quicksort: An Example

The Partitioning Phase:

pivot ↓

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
		↑ L (*pause*)											↑ R		

Quicksort: An Example

The Partitioning Phase:

pivot ↓															
6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
		↑ L (*pause*)										↑ R			

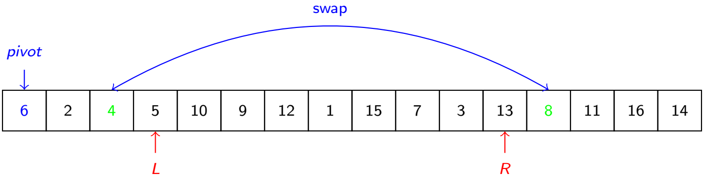
The Partitioning Phase:

pivot ↓

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14

↑ *L* (*pause*) ↑ *R* (*pause*)

The Partitioning Phase:



Quicksort: An Example

The Partitioning Phase:

pivot															
6	2	4	5	10	9	12	1	15	7	3	13	8	11	16	14
			L								R				

Quicksort: An Example

The Partitioning Phase:

pivot ↓

6	2	4	5	10	9	12	1	15	7	3	13	8	11	16	14

↑ L (*pause*) ↑ R

Quicksort: An Example

The Partitioning Phase:

pivot

6	2	4	5	10	9	12	1	15	7	3	13	8	11	16	14

L (*pause*) R

Quicksort: An Example

The Partitioning Phase:

pivot ↓

6	2	4	5	10	9	12	1	15	7	3	13	8	11	16	14
↑ pivot				↑ *L* (*pause*)						↑ *R* (*pause*)					

Quicksort: An Example

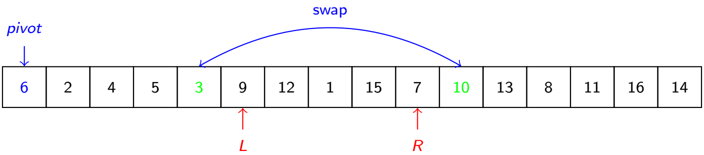
The Partitioning Phase:

pivot

6	2	4	5	10	9	12	1	15	7	3	13	8	11	16	14

L (*pause*) R (*pause*)

The Partitioning Phase:



Quicksort: An Example

The Partitioning Phase:

pivot ↓															
6	2	4	5	3	9	12	1	15	7	10	13	8	11	16	14
					↑ *L (pause)*				↑ *R*						

The Partitioning Phase:

pivot

6	2	4	5	3	9	12	1	15	7	10	13	8	11	16	14

L (pause) *R*

Quicksort: An Example

The Partitioning Phase:

pivot ↓															
6	2	4	5	3	9	12	1	15	7	10	13	8	11	16	14
					↑ *L* (*pause*)			↑ *R*							

Quicksort: An Example

The Partitioning Phase:

pivot

6	2	4	5	3	9	12	1	15	7	10	13	8	11	16	14

L (*pause*) *R* (*pause*)

The Partitioning Phase:

pivot swap

6	2	4	5	3	1	12	9	15	7	10	13	8	11	16	14

$L = R$

Quicksort: An Example

The Partitioning Phase:

swap

pivot

1	2	4	5	3	6	12	9	15	7	10	13	8	11	16	14

$L = R$

Recursive Phase:

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14

Quicksort: An Example

Recursive Phase:

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14

Quicksort: An Example

Recursive Phase:

1	2	4	5	3	6	12	9	15	7	10	13	8	11	16	14

Quicksort: An Example

Recursive Phase:

1	2	4	5	3	6	12	9	15	7	10	13	8	11	16	14

Quicksort: An Example

Recursive Phase:

1	2	4	5	3	6	12	9	15	7	10	13	8	11	16	14

Quicksort: An Example

Recursive Phase:

1	2	3	4	5	6	12	9	15	7	10	13	8	11	16	14

Note: When a single number appears between two pivots it is obviously in the right position.

Quicksort: An Example

Recursive Phase:

1	2	3	4	5	6	8	9	11	7	10	12	13	15	16	14

Quicksort: An Example

Recursive Phase:

1	2	3	4	5	6	7	8	11	9	10	12	13	15	16	14

Quicksort: An Example

Recursive Phase:

1	2	3	4	5	6	7	8	11	9	10	12	13	15	16	14

Recursive Phase:

1	2	3	4	5	6	7	8	10	9	11	12	13	15	16	14

Recursive Phase:

1	2	3	4	5	6	7	8	9	10	11	12	13	15	16	14

Quicksort: An Example

Recursive Phase:

1	2	3	4	5	6	7	8	9	10	11	12	13	15	16	14

Quicksort: An Example

Recursive Phase:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Quicksort: An Example

Recursive Phase:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Home Work: Write the Quicksort algorithm and implement it in C.

Correctness

- Guaranteed by the following *loop invariant*:

  "At step k of the algorithm, *pivot* $\geq x_i$ for all i such that $i < L$, and *pivot* $< X_j$ for all j such that $j > R$".

Correctness

- Guaranteed by the following *loop invariant*:

  "At step k of the algorithm, $pivot \geq x_i$ for all i such that $i < L$, and $pivot < X_j$ for all j such that $j > R$".

  Home Work: Prove it using mathematical induction.

Correctness

- Guaranteed by the following *loop invariant*:

  "At step k of the algorithm, $pivot \geq x_i$ for all i such that $i < L$, and $pivot < X_j$ for all j such that $j > R$".

  Home Work: Prove it using mathematical induction.

- **Termination:** When $L = R$.

Choosing a Good Pivot

- Divide-and-conquer algorithms work best when the parts have equal sizes.
- $\therefore$ the closer the pivot is to the middle, the faster the algorithm.
- It is possible to find the median of the sequence (using the *Median finding algorithm*), but it is not worth the effort.
- In fact, choosing a uniform random element suffices.
- If the sequence is in a *uniformly random order*, then we might as well choose the first element as the pivot.

Cost Analysis

- Running time: Depends on the input sequence and pivot.
- If the pivot always partitions the list into two equal parts, then

$$\begin{aligned} T(n) &= 2T(n/2) + \mathcal{O}(n), \quad T(2) = 1, \\ \Rightarrow T(n) &= \mathcal{O}(n \log n). \end{aligned}$$

Cost Analysis

- Running time: Depends on the input sequence and pivot.
- If the pivot always partitions the list into two equal parts, then

$$\begin{aligned} T(n) &= 2T(n/2) + \mathcal{O}(n), \quad T(2) = 1, \\ \Rightarrow T(n) &= \mathcal{O}(n \log n). \end{aligned}$$

- But we can get $\mathcal{O}(n \log n)$ even under much *weaker conditions*!

Cost Analysis

- Running time: Depends on the input sequence and pivot.
- If the pivot always partitions the list into two equal parts, then

$$\begin{aligned} T(n) &= 2T(n/2) + \mathcal{O}(n), \quad T(2) = 1, \\ \Rightarrow T(n) &= \mathcal{O}(n \log n). \end{aligned}$$

- But we can get $\mathcal{O}(n \log n)$ even under much *weaker conditions*!
- However, if the pivot is very close to one side of the sequence, then the running time is much higher.

Cost Analysis

- Running time: Depends on the input sequence and pivot.
- If the pivot always partitions the list into two equal parts, then

$$\begin{aligned} T(n) &= 2T(n/2) + \mathcal{O}(n), \quad T(2) = 1, \\ \Rightarrow T(n) &= \mathcal{O}(n \log n). \end{aligned}$$

- But we can get $\mathcal{O}(n \log n)$ even under much *weaker conditions*!
- However, if the pivot is very close to one side of the sequence, then the running time is much higher.

  Example:

  - If the sequence is already sorted.
  - **Time Complexity:** $\mathcal{O}(n^2)$.

Cost Analysis (Cont.)

- ...
- The quadratic worst case for (almost) sorted sequences can be eliminated
  - by comparing the first, last, and middle elements,
  - and then taking their median (the second largest) as the pivot.

Cost Analysis (Cont.)

- ...
- The quadratic worst case for (almost) sorted sequences can be eliminated
  - by comparing the first, last, and middle elements,
  - and then taking their median (the second largest) as the pivot.
- **Safer method:** Choose the pivot randomly from among the elements in the sequence.
- **Worst-case complexity:** $\mathcal{O}(n^2)$ (since there is still a chance that the pivot is the smallest element in the sequence.)

Cost Analysis (Cont.)

- ...
- The quadratic worst case for (almost) sorted sequences can be eliminated
  - by comparing the first, last, and middle elements,
  - and then taking their median (the second largest) as the pivot.
- **Safer method:** Choose the pivot randomly from among the elements in the sequence.
- **Worst-case complexity:** $\mathcal{O}(n^2)$ (since there is still a chance that the pivot is the smallest element in the sequence.)
- However, the likelihood that this worst case occur is very small.

Average-case Complexity

Given a sequence $x_1, \ldots, x_n$, assume that each of the x_i has the same probability of being selected as the pivot.

Average-case Complexity

Given a sequence $x_1, \ldots, x_n$, assume that each of the x_i has the same probability of being selected as the pivot.

Running time when i^{th} smallest element is the pivot:

$$T(n) = \underbrace{n-1}_{\text{partitioning}} + T(i-1) + T(n-i).$$

Average-case Complexity

Given a sequence $x_1, \ldots, x_n$, assume that each of the x_i has the same probability of being selected as the pivot.

Running time when i^{th} smallest element is the pivot:

$$T(n) = \underbrace{n-1}_{\text{partitioning}} + T(i-1) + T(n-i).$$

The average-case time complexity is then given by

$$\begin{aligned} T(n) &= n - 1 + \frac{1}{n}\sum_{i=1}^{n}(T(i-1) + T(n-i)) \\ &= n - 1 + \frac{2}{n}\sum_{i=0}^{n-1} T(i) \quad \text{[Full history recurrence]} \end{aligned}$$

Full History Recurrences

Full History Recurrence

Definition

A full-history recurrence relation is one that depends on all the previous values of the function, not just on a few of them.

A Simplest Full-history Recurrence Relation

Consider

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

where c is a constant and $T(1)$ is given.

A Simplest Full-history Recurrence Relation

Consider

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

where c is a constant and $T(1)$ is given.

Solution:

- We use a method that cancels most of the intermediate terms.

A Simplest Full-history Recurrence Relation

Consider

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

where c is a constant and $T(1)$ is given.

Solution:

- We use a method that cancels most of the intermediate terms.
- Sometimes called *elimination of history*.

A Simplest Full-history Recurrence Relation

Consider

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

where c is a constant and $T(1)$ is given.

Solution:

- We use a method that cancels most of the intermediate terms.
- Sometimes called *elimination of history*.
- Compare $T(n+1)$ with $T(n)$ and subtract to get

$$T(n+1) - T(n) = T(n) \;\Rightarrow\; T(n+1) = 2T(n) \;\Rightarrow\; T(n+1) = T(1)2^n.$$

A Simplest Full-history Recurrence Relation

Consider

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

where c is a constant and $T(1)$ is given.

Solution:

- We use a method that cancels most of the intermediate terms.
- Sometimes called *elimination of history*.
- Compare $T(n+1)$ with $T(n)$ and subtract to get

  $$T(n+1) - T(n) = T(n) \;\Rightarrow\; T(n+1) = 2T(n) \;\Rightarrow\; T(n+1) = T(1)2^n.$$

- **Note:**
  - The claim is true for $T(1)$, $\because$ $T(1) = T(0+1) = T(1)2^0 = T(1)$.

A Simplest Full-history Recurrence Relation

Consider

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

where c is a constant and $T(1)$ is given.

Solution:

- We use a method that cancels most of the intermediate terms.
- Sometimes called *elimination of history*.
- Compare $T(n+1)$ with $T(n)$ and subtract to get

  $$T(n+1) - T(n) = T(n) \Rightarrow T(n+1) = 2T(n) \Rightarrow T(n+1) = T(1)2^n.$$

- **Note:**
  - The claim is true for $T(1)$, $\because$ $T(1) = T(0+1) = T(1)2^0 = T(1)$.
  - **Induction step:** If the claim is true for $T(n)$, then

    $$T(n+1) = 2T(n) = T(1)2^n.$$

Correct?

A Simplest Full-history Recurrence Relation (Cont.)

Correct? No!!

A Simplest Full-history Recurrence Relation (Cont.)

Correct? No!!

Example: Set $T(1) = 1$ and $c = 5$, s.t., $T(2) = 6 \neq 2T(1) = 2$.

Correct? No!!

Example: Set $T(1) = 1$ and $c = 5$, s.t., $T(2) = 6 \neq 2T(1) = 2$.

Reason:

- This is an example of carelessly going through an induction proof *ignoring the base case*.

A Simplest Full-history Recurrence Relation (Cont.)

Correct? No!!

Example: Set $T(1) = 1$ and $c = 5$, s.t., $T(2) = 6 \neq 2T(1) = 2$.

Reason:

- This is an example of carelessly going through an induction proof *ignoring the base case*.
- **Note:** The proof does not work for $T(2)$, since $T(2) - T(1) = c$ may not be equal to $T(1)$.

Correct? No!!

Example: Set $T(1) = 1$ and $c = 5$, s.t., $T(2) = 6 \neq 2T(1) = 2$.

Reason:

- This is an example of carelessly going through an induction proof *ignoring the base case*.
- **Note:** The proof does not work for $T(2)$, since $T(2) - T(1) = c$ may not be equal to $T(1)$.
- **Warning:** *One should be very suspicious when a parameter (c in this case) that appears in the expression does not appear in the final solution.*

A Simplest Full-history Recurrence Relation (Cont.)

Correct? No!!

Example: Set $T(1) = 1$ and $c = 5$, s.t., $T(2) = 6 \neq 2T(1) = 2$.

Reason:

- This is an example of carelessly going through an induction proof *ignoring the base case*.
- **Note:** The proof does not work for $T(2)$, since $T(2) - T(1) = c$ may not be equal to $T(1)$.
- **Warning:** *One should be very suspicious when a parameter (c in this case) that appears in the expression does not appear in the final solution.*
- **Correct Solution:** Note that $T(2) = T(1) + c$ (by definition), and that the proof above is correct for all $n \geq 2$.

Correct? No!!

Example: Set $T(1) = 1$ and $c = 5$, s.t., $T(2) = 6 \neq 2T(1) = 2$.

Reason:

- This is an example of carelessly going through an induction proof *ignoring the base case*.
- **Note:** The proof does not work for $T(2)$, since $T(2) - T(1) = c$ may not be equal to $T(1)$.
- **Warning:** *One should be very suspicious when a parameter (c in this case) that appears in the expression does not appear in the final solution.*
- **Correct Solution:** Note that $T(2) = T(1) + c$ (by definition), and that the proof above is correct for all $n \geq 2$.
- $\therefore T(n+1) = (T(1) + c)2^{n-1}$.

A Not So Simple Full-history Recurrence Relation

$$T(n) = n - 1 + \frac{2}{n} \sum_{i=1}^{n-1} T(i), \text{ (for } n \geq 2); \; T(1) = 0.$$

A Not So Simple Full-history Recurrence Relation

$$T(n) = n - 1 + \frac{2}{n} \sum_{i=1}^{n-1} T(i), \text{ (for } n \geq 2); \; T(1) = 0.$$

- **Idea:** Use the shifting and canceling terms technique, s.t., most of the $T(i)$ terms gets canceled out.

A Not So Simple Full-history Recurrence Relation (Cont.)

Multiplying both sides by n, we get:

$$\begin{aligned} nT(n) &= n(n-1) + 2\sum_{i=1}^{n-1} T(i), \\ (n+1)T(n+1) &= n(n+1) + 2\sum_{i=1}^{n} T(i). \end{aligned}$$

A Not So Simple Full-history Recurrence Relation (Cont.)

Multiplying both sides by n, we get:

$$
\begin{aligned}
nT(n) &= n(n-1) + 2\sum_{i=1}^{n-1} T(i), \\
(n+1)T(n+1) &= n(n+1) + 2\sum_{i=1}^{n} T(i).
\end{aligned}
$$

Therefore,

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

A Not So Simple Full-history Recurrence Relation (Cont.)

Multiplying both sides by n, we get:

$$\begin{aligned} nT(n) &= n(n-1) + 2\sum_{i=1}^{n-1} T(i), \\ (n+1)T(n+1) &= n(n+1) + 2\sum_{i=1}^{n} T(i). \end{aligned}$$

Therefore,

$$\begin{aligned} (n+1)T(n+1) - nT(n) &= n(n+1) - n(n-1) + 2T(n) \\ &= 2n + 2T(n) \end{aligned}$$

A Not So Simple Full-history Recurrence Relation (Cont.)

Multiplying both sides by n, we get:

$$\begin{aligned} nT(n) &= n(n-1) + 2\sum_{i=1}^{n-1} T(i), \\ (n+1)T(n+1) &= n(n+1) + 2\sum_{i=1}^{n} T(i). \end{aligned}$$

Therefore,

$$\begin{aligned} (n+1)T(n+1) - nT(n) &= n(n+1) - n(n-1) + 2T(n) \\ &= 2n + 2T(n) \\ \Rightarrow \quad T(n+1) &= \frac{n+2}{n+1}T(n) + \frac{2n}{n+1} \end{aligned}$$

A Not So Simple Full-history Recurrence Relation (Cont.)

Multiplying both sides by n, we get:

$$\begin{aligned} nT(n) &= n(n-1) + 2\sum_{i=1}^{n-1} T(i), \\ (n+1)T(n+1) &= n(n+1) + 2\sum_{i=1}^{n} T(i). \end{aligned}$$

Therefore,

$$\begin{aligned} (n+1)T(n+1) - nT(n) &= n(n+1) - n(n-1) + 2T(n) \\ &= 2n + 2T(n) \\ \Rightarrow \ T(n+1) &= \frac{n+2}{n+1}T(n) + \frac{2n}{n+1} \\ &\leq \frac{n+2}{n+1}T(n) + 2 \quad \text{(A close approx.)} \end{aligned}$$

A Not So Simple Full-history Recurrence Relation (Cont.)

Expanding, we get

$$T(n) \leq 2 + \frac{n+1}{n}\left[2 + \frac{n}{n-1}\left[2 + \frac{n-1}{n-2}\left[\cdots \frac{4}{3}\right]\right]\right]$$

A Not So Simple Full-history Recurrence Relation (Cont.)

Expanding, we get

$$
\begin{aligned}
T(n) &\leq 2 + \frac{n+1}{n}\left[2 + \frac{n}{n-1}\left[2 + \frac{n-1}{n-2}\left[\cdots \frac{4}{3}\right]\right]\right] \\
&= 2\left[1 + \frac{n+1}{n} + \frac{n+1}{n} \cdot \frac{n}{n-1} + \frac{n+1}{n} \cdot \frac{n}{n-1} \cdot \frac{n-1}{n-2} + \right. \\
&\qquad \left. \cdots + \frac{n+1}{n} \cdot \frac{n}{n-1} \cdot \frac{n-1}{n-2} \cdots \frac{4}{3}\right]
\end{aligned}
$$

A Not So Simple Full-history Recurrence Relation (Cont.)

Expanding, we get

$$\begin{aligned}
T(n) &\leq 2 + \frac{n+1}{n}\left[2 + \frac{n}{n-1}\left[2 + \frac{n-1}{n-2}\left[\cdots\frac{4}{3}\right]\right]\right] \\
&= 2\left[1 + \frac{n+1}{n} + \frac{n+1}{n}\cdot\frac{n}{n-1} + \frac{n+1}{n}\cdot\frac{n}{n-1}\cdot\frac{n-1}{n-2} + \right. \\
&\qquad \left. \cdots + \frac{n+1}{n}\cdot\frac{n}{n-1}\cdot\frac{n-1}{n-2}\cdots\frac{4}{3}\right] \\
&= 2\left[1 + \frac{n+1}{n} + \frac{n+1}{n-1} + \frac{n+1}{n-2} + \cdots + \frac{n+1}{3}\right]
\end{aligned}$$

Expanding, we get

$$\begin{aligned}
T(n) &\leq 2 + \frac{n+1}{n}\left[2 + \frac{n}{n-1}\left[2 + \frac{n-1}{n-2}\left[\cdots\frac{4}{3}\right]\right]\right] \\
&= 2\left[1 + \frac{n+1}{n} + \frac{n+1}{n}\cdot\frac{n}{n-1} + \frac{n+1}{n}\cdot\frac{n}{n-1}\cdot\frac{n-1}{n-2} + \right. \\
&\quad \left. \cdots + \frac{n+1}{n}\cdot\frac{n}{n-1}\cdot\frac{n-1}{n-2}\cdots\frac{4}{3}\right] \\
&= 2\left[1 + \frac{n+1}{n} + \frac{n+1}{n-1} + \frac{n+1}{n-2} + \cdots + \frac{n+1}{3}\right] \\
&= 2(n+1)\left[\frac{1}{n+1} + \frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + \cdots + \frac{1}{3}\right]
\end{aligned}$$

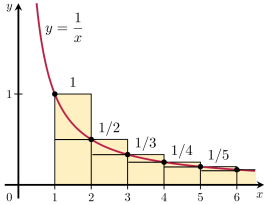
A Not So Simple Full-history Recurrence Relation (Cont.)

Expanding, we get

$$\begin{aligned}
T(n) &\leq 2 + \frac{n+1}{n}\left[2 + \frac{n}{n-1}\left[2 + \frac{n-1}{n-2}\left[\cdots \frac{4}{3}\right]\right]\right] \\
&= 2\left[1 + \frac{n+1}{n} + \frac{n+1}{n}\cdot\frac{n}{n-1} + \frac{n+1}{n}\cdot\frac{n}{n-1}\cdot\frac{n-1}{n-2} + \right. \\
&\quad \left. \cdots + \frac{n+1}{n}\cdot\frac{n}{n-1}\cdot\frac{n-1}{n-2}\cdots\frac{4}{3}\right] \\
&= 2\left[1 + \frac{n+1}{n} + \frac{n+1}{n-1} + \frac{n+1}{n-2} + \cdots + \frac{n+1}{3}\right] \\
&= 2(n+1)\left[\frac{1}{n+1} + \frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + \cdots + \frac{1}{3}\right] \\
&= 2(n+1)(H(n+1) - 1.5);
\end{aligned}$$

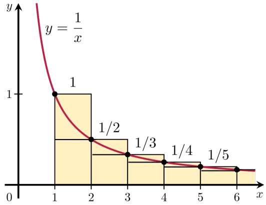
where $H(n) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}$ is the Harmonic series.

Harmonic Series Approximation



$$H(n) = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \quad > \quad \int_1^{n+1} \frac{dx}{x} = \ln(n+1) \quad \text{and}$$

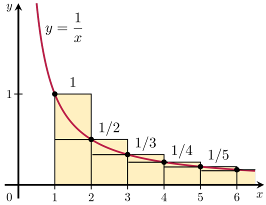
Harmonic Series Approximation



$$H(n) = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \quad > \quad \int_1^{n+1} \frac{dx}{x} = \ln(n+1) \quad \text{and}$$

$$H(n) = 1 + \left(\frac{1}{2} + \cdots + \frac{1}{n}\right) \quad < \quad 1 + \int_1^{n} \frac{dx}{x} = 1 + \ln(n).$$

Harmonic Series Approximation



$$H(n) = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \quad > \quad \int_1^{n+1} \frac{dx}{x} = \ln(n+1) \quad \text{and}$$

$$H(n) = 1 + \left(\frac{1}{2} + \cdots + \frac{1}{n}\right) \quad < \quad 1 + \int_1^{n} \frac{dx}{x} = 1 + \ln(n).$$

Combining: $\ln(n+1) < H(n) < 1 + \ln(n)$.

Harmonic Series Approximation (Cont.)

Let $\delta_n = H(n) - \ln n$.

Harmonic Series Approximation (Cont.)

Let $\delta_n = H(n) - \ln n$.
Then,

$$\ln(1 + 1/n) < H(n) - \ln n < 1, \text{ and}$$

Harmonic Series Approximation (Cont.)

Let $\delta_n = H(n) - \ln n$.
Then,

$$\ln(1 + 1/n) < H(n) - \ln n < 1, \text{ and}$$

$$\delta_n - \delta_{n+1} = (H(n) - \ln n) - (H(n+1) - \ln(n+1))$$

Harmonic Series Approximation (Cont.)

Let $\delta_n = H(n) - \ln n$.
Then,

$$\ln(1 + 1/n) < H(n) - \ln n < 1, \text{ and}$$

$$\begin{aligned} \delta_n - \delta_{n+1} &= (H(n) - \ln n) - (H(n+1) - \ln(n+1)) \\ &= \ln(n+1) - \ln n - \frac{1}{n+1} \end{aligned}$$

Harmonic Series Approximation (Cont.)

Let $\delta_n = H(n) - \ln n$.
Then,

$$\ln(1 + 1/n) < H(n) - \ln n < 1, \text{ and}$$

$$\begin{aligned}
\delta_n - \delta_{n+1} &= (H(n) - \ln n) - (H(n+1) - \ln(n+1)) \\
&= \ln(n+1) - \ln n - \frac{1}{n+1} \\
&= \int_n^{n+1} \frac{dx}{x} - \frac{1}{n+1} > 0 \quad \text{(See the picture!)},
\end{aligned}$$

i.e., δ_n is monotone decreasing.

Harmonic Series Approximation (Cont.)

Let $\delta_n = H(n) - \ln n$.
Then,

$$\ln(1 + 1/n) < H(n) - \ln n < 1, \text{ and}$$

$$\begin{aligned}
\delta_n - \delta_{n+1} &= (H(n) - \ln n) - (H(n+1) - \ln(n+1)) \\
&= \ln(n+1) - \ln n - \frac{1}{n+1} \\
&= \int_n^{n+1} \frac{dx}{x} - \frac{1}{n+1} > 0 \quad \text{(See the picture!)},
\end{aligned}$$

i.e., δ_n is monotone decreasing. Therefore δ_n converges and let

$$\gamma = \lim_{n\to\infty} \delta_n = \lim_{n\to\infty} (H(n) - \ln n).$$

Harmonic Series Approximation (Cont.)

Let $\delta_n = H(n) - \ln n$.
Then,

$$\ln(1 + 1/n) < H(n) - \ln n < 1, \text{ and}$$

$$\begin{aligned}
\delta_n - \delta_{n+1} &= (H(n) - \ln n) - (H(n+1) - \ln(n+1)) \\
&= \ln(n+1) - \ln n - \frac{1}{n+1} \\
&= \int_n^{n+1} \frac{dx}{x} - \frac{1}{n+1} > 0 \quad \text{(See the picture!)},
\end{aligned}$$

i.e., δ_n is monotone decreasing. Therefore δ_n converges and let

$$\gamma = \lim_{n\to\infty} \delta_n = \lim_{n\to\infty} (H(n) - \ln n).$$

$\gamma \approx 0.5772$ is called the **Euler constant** (Euler, 1735).

Harmonic Series Approximation (Cont.)

Let $\delta_n = H(n) - \ln n$.
Then,

$$\ln(1 + 1/n) < H(n) - \ln n < 1, \text{ and}$$

$$\begin{aligned}
\delta_n - \delta_{n+1} &= (H(n) - \ln n) - (H(n+1) - \ln(n+1)) \\
&= \ln(n+1) - \ln n - \frac{1}{n+1} \\
&= \int_n^{n+1} \frac{dx}{x} - \frac{1}{n+1} > 0 \quad \text{(See the picture!)},
\end{aligned}$$

i.e., δ_n is monotone decreasing. Therefore δ_n converges and let

$$\gamma = \lim_{n\to\infty} \delta_n = \lim_{n\to\infty} (H(n) - \ln n).$$

$\gamma \approx 0.5772$ is called the **Euler constant** (Euler, 1735).
$\therefore\ H(n) \approx \ln n + \gamma$.

A Not So Simple Full-history Recurrence Relation (Cont.)

Using the Harmonic series approximation, we get

$$T(n) \quad \leq \quad 2(n+1)(H(n+1) - 1.5)$$

Using the Harmonic series approximation, we get

$$\begin{aligned} T(n) &\leq 2(n+1)(H(n+1) - 1.5) \\ &\approx 2(n+1)(\ln n + \gamma - 1.5) + \mathcal{O}(1) \end{aligned}$$

A Not So Simple Full-history Recurrence Relation (Cont.)

Using the Harmonic series approximation, we get

$$\begin{aligned} T(n) &\leq 2(n+1)(H(n+1) - 1.5) \\ &\approx 2(n+1)(\ln n + \gamma - 1.5) + \mathcal{O}(1) \\ &= \mathcal{O}(n \log n). \end{aligned}$$

A Not So Simple Full-history Recurrence Relation (Cont.)

Using the Harmonic series approximation, we get

$$\begin{aligned} T(n) &\leq 2(n+1)(H(n+1) - 1.5) \\ &\approx 2(n+1)(\ln n + \gamma - 1.5) + \mathcal{O}(1) \\ &= \mathcal{O}(n \log n). \end{aligned}$$

$\therefore$ **Quicksort is indeed quick on the average!!**

Few Remarks

- In practice, quicksort is very fast, so it well deserves it’s name.

Few Remarks

- In practice, quicksort is very fast, so it well deserves it's name.
- **Reason:**
  - Many elements are compared against the same pivot element.
  - $\therefore$ the pivot can be stored in a register.
  - It is an in-place algorithm.

Few Remarks

- In practice, quicksort is very fast, so it well deserves it's name.
- **Reason:**
  - Many elements are compared against the same pivot element.
  - $\therefore$ the pivot can be stored in a register.
  - It is an in-place algorithm.
- **Improvement:** By choosing the base of the induction wisely.

Few Remarks

- In practice, quicksort is very fast, so it well deserves it's name.
- **Reason:**
  - Many elements are compared against the same pivot element.
  - $\therefore$ the pivot can be stored in a register.
  - It is an in-place algorithm.
- **Improvement:** By choosing the base of the induction wisely.
  - The idea is to start the induction not always from 1.
  - Use, simple sorting techniques, like insertion sort or selection sort for small sequences.
  - **Note:** The efficiency of quicksort shows only for large sequences.
  - Define the base case for quicksort to be of size larger than 1 (10 to 20 is a good size).
  - Handle the base case by insertion sort or selection sort.

Books Consulted

1. Chapter 6, Section 4.3 & 4.4 of *Introduction to Algorithms: A Creative Approach* by Udi Manber.

Thank You for your kind attention!