2021-2022

Data Structures and Algorithms (II) – Sorting Algorithms

Dr. Dapeng Dong

Quicksort

- The quicksort algorithm is also a divide-and-conquer recursive algorithm, which has a worst-case running time of $\boldsymbol{\theta}(\boldsymbol{N^2})$ on an input array of $\boldsymbol{N}$ values.

- It has an average-case running time of $\boldsymbol{\theta}(\boldsymbol{N}\mathbf{lg}\boldsymbol{N})$, but it is remarkably efficient on the average because the constant factors in the $\theta(N\lg N)$ notation are quite small (highly optimised inner loop).

- It has the advantage of sorting in place, i.e., transforms input without using auxiliary data structures.

Quicksort Process

```
QUICKSORT(data, $l$, $r$)
    if $l$ < $r$
Divide      p = PARITITION(data, $l$, $r$)
Conquer     QUICKSORT(data, $l$, $p$-1)
            QUICKSORT(data, $p$+1, $r$)
```

Given an input array $\boldsymbol{data[l\ \ldots\ r]}$:

- **Divide:**
  - Select an element in the array. The selected element is known as the ***pivot***.
  - Partition the array into two subarrays based on the the pivot selected, $\boldsymbol{data[l\ \ldots p\ -1]}$ and $\boldsymbol{data[p\ +1\ \ldots\ r]}$, such that each element of $\boldsymbol{data[l\ \ldots p\ -1]}$ is less than or equal to the pivot, and each element of $\boldsymbol{data[p\ +1\ \ldots\ r]}$ is greater than or equal to the pivot.
- **Conquer:**
  - Sort the two subarrays by recursive calls to quicksort.
- ***Combine:***
  - No combine process is needed as the two subarrays are already sorted and the partition scheme ensures that the two subarrays are in order.

$\boldsymbol{l}$: left

$\boldsymbol{r}$: right

$\boldsymbol{p}$: partition point

Partitioning

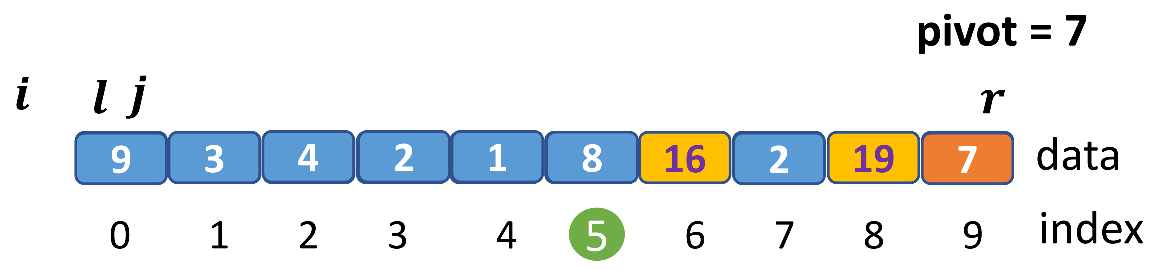
- The key to the algorithm is the partition procedure, which rearranges subarrays in place.

l: left
r: right

```
partition(data, $l$, $r$)
    pivot = data[ $r$ ]
    $i$ = $l$ - 1
    for $j$ = $l$ to $r$ - 1
        if data[ $j$ ] $\leq$ pivot
            $i$++
            SWAP(data[ $i$ ], data[ $j$ ])

    SWAP(data[ $i + 1$ ], data[ $r$ ])
    return $i + 1$
```

Partition procedure in principle



Picking a Right *Pivot* is Important (The Worst-Case Scenario) 1

1 Similar to merge sort, quick sort is recursive. It is clear that the partition procedure on an array of N elements takes a linear time to complete, thus

$$T(N) = \begin{cases} c & \mathit{if}\ N = 1, \\ T(i) + T(N - i - 1) + cN & \mathit{if}\ N > 1, \end{cases}$$

partition 1 partition 2 the pivot

```
QUICKSORT(data, $l$, $r$)
    if $l < r$
Divide      $p$ = PARITITION(data, $l$, $r$)
Conquer     QUICKSORT(data, $l$, $p$-1)
            QUICKSORT(data, $p$+1, $r$)
```

```
PATITION(data, $l$, $r$)
    pivot = data[ $r$ ]
    $i$ = $l$ - 1
    for $j$ = $l$ to $r$ - 1
        if data[ $j$ ] $\leq$ pivot
            $i$++
            SWAP(data[ $i$ ], data[ $j$ ])

    SWAP(data[ $i + 1$ ], data[ $r$ ])
    return $i + 1$
```

2 In the worst-case scenario, the partition is unbalanced. For example when the pivot is the smallest element in the array at all time, then $i = 0$, i.e., the first partition contains no element, whereas the second partition contains the $N - 1$ element (excluding the pivot), thus

$$T(N) = T(0) + T(N - 1) + cN$$

Picking a Right *Pivot* is Important (The Worst-Case Scenario) 2

$$T(N) = T(0) + T(N-1) + cN$$

3 Telescoping,

$$T(N) = T(N-1) + cN$$

$$T(N-1) = T(N-2) + c(N-1)$$

$$T(N-2) = T(N-3) + c(N-2)$$

$$T(2) = T(1) + c(1)$$

A The n^{th} partial sums of a divergent series (the triangular number)

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

3 Collecting the terms,

$$T(N) = T(1) + c\,\{N + (N-1) + (N-2) + \cdots + 3 + 2\}$$

$$= T(1) + c\sum_{i=2}^{N} i$$

4 Given $T(1) = 1$, i.e., the algorithm takes a constant time to sort when only one element is in the array, thus,

$$T(N) = \frac{1}{2}cN^2 + \frac{1}{2}cN + 1$$

5 Ignoring the lower terms and constant,

$$T(N) = \theta(N^2)$$

Picking a Right *Pivot* is Important (The Best-Case Scenario) 1

$$T(N) = \begin{cases} c & if\ N = 1, \\ T(i) + T(N-i-1) + cN & if\ N > 1, \end{cases}$$

1 In another case, if the selection of the pivot always leads the partition procedure to split subarrays evenly, the two subarrays are each half the size of the original (to simplify the analysis), thus,

$$T(N) = 2T\left(\frac{N}{2}\right) + cN$$

2 Dividing both sides by N, so that it telescopes,

$$\frac{T(N)}{N} = \frac{T(N/2)}{N/2} + c$$

3 Telescoping,

$$\frac{T(N)}{N} = \frac{T(N/2)}{N/2} + c$$

$$\frac{T(N/2)}{N/2} = \frac{T(N/4)}{N/4} + c$$

$$\frac{T(N/4)}{N/4} = \frac{T(N/8)}{N/8} + c$$

$$\vdots$$

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

Picking a Right *Pivot* is Important (The Best-Case Scenario) 2

5

$$\langle N, \frac{N}{2}, \frac{N}{4}, \frac{N}{8}, \dots, 4, 2 \rangle \Leftarrow \left\{ \begin{aligned} \frac{T(N)}{N} &= \frac{T(N/2)}{N/2} + c \\ \frac{T(N/2)}{N/2} &= \frac{T(N/4)}{N/4} + c \\ \frac{T(N/4)}{N/4} &= \frac{T(N/8)}{N/8} + c \\ &\vdots \\ \frac{T(2)}{2} &= \frac{T(1)}{1} + c \end{aligned} \right.$$

4 Collecting the terms,

Collecting the terms and rearrange,

$$\frac{T(N)}{N} = \frac{T(1)}{1} + \sum_{i=1}^{k} c_i$$

5.1 Assuming N is a power of 2, let $N = 2^k$

$$\langle N, \frac{N}{2}, \frac{N}{4}, \frac{N}{8}, \dots, 4, 2 \rangle \implies \langle 2^k, 2^{k-1}, 2^{k-2}, \dots, 2^2, 2^1 \rangle$$

Thus, k indicates the number of c in 4

5.2 Taking logarithm to the base 2 on both sides,

$$N = 2^k \implies \lg N = k$$

6 Given that $T(1) = 1$,

$$T(N) = cN\lg N + N$$

6.1 Ignoring the lower-term and constant,

$$T(N) = \theta(N\lg N)$$

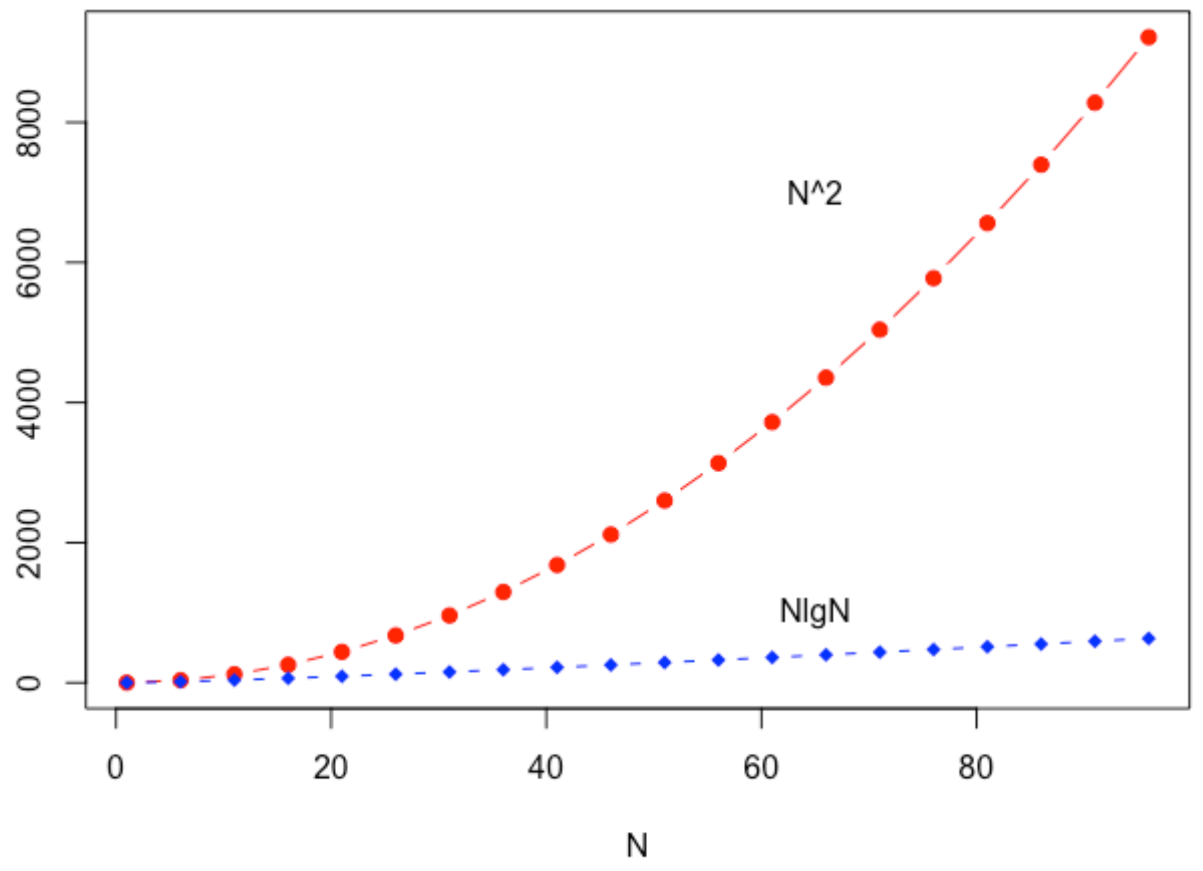
Picking a Right *Pivot* is Important

Worst-case running time:

$$T(N) = \theta(N^2)$$

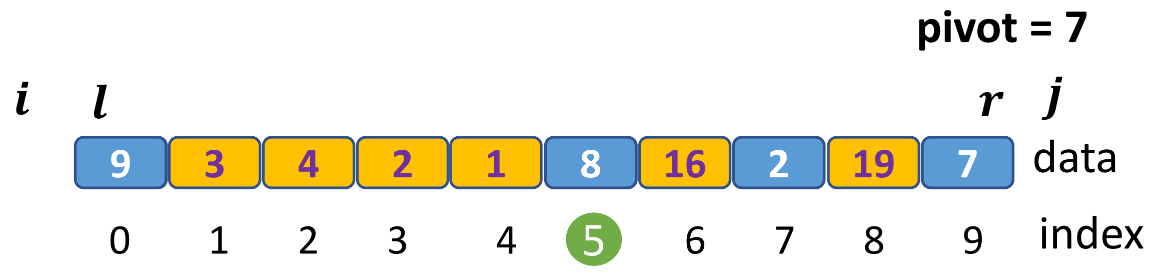
Best-case running time:

$$T(N) = \theta(N\lg N)$$



Partitioning Strategy (Hoare)

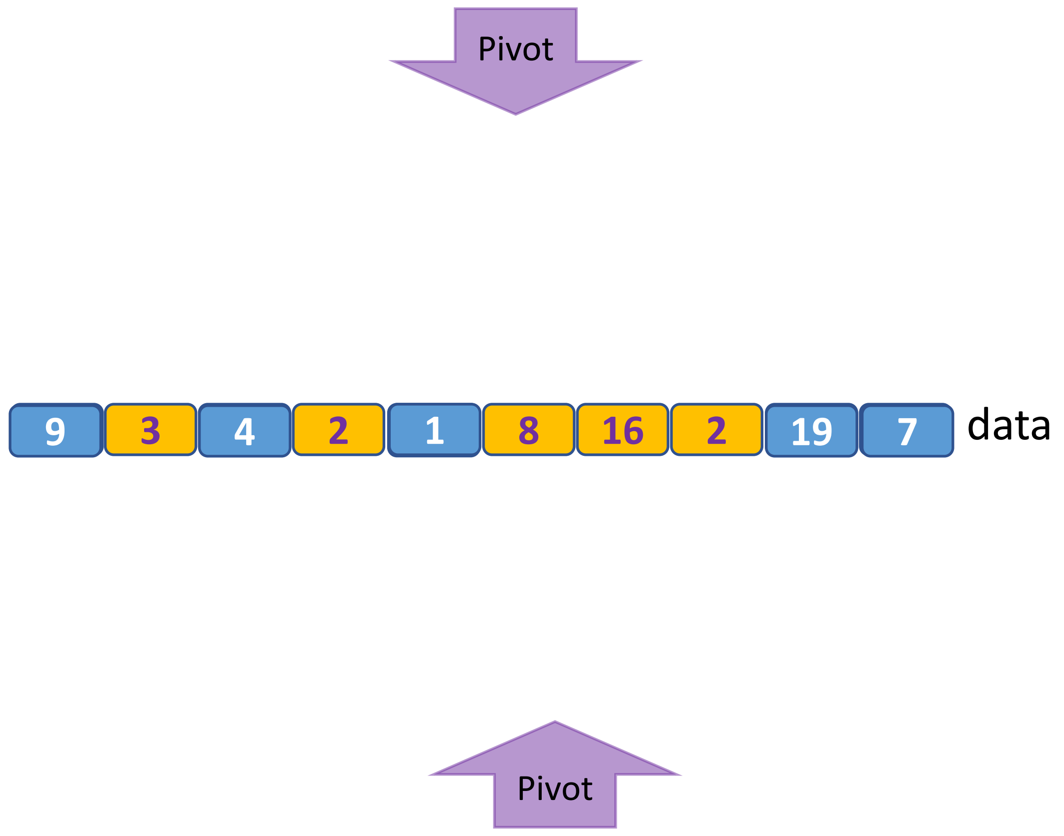
```
HOARE-PATITION(data, $l$, $r$)
    pivot = data[ $r$ ]
    $i = l - 1$
    $j = r + 1$
    while TRUE
        repeat
            $j = j - 1$
        until data[ $j$ ] ≤ pivot
        repeat
            $i = i + 1$
        until data[ $i$ ] ≥ pivot
        if $i < j$
            SWAP(data[ $i$ ], data[ $j$ ])
        else return $j$
```



Partitioning Strategy (Median-of-Three)

- Scheme 1:
  - Randomly choose three elements and then use the median of the three as the pivot

- Scheme 2:
  - Take the left, right and centre values, and then use the median of the three as the pivot



Average-Case Analysis with a Random Partition Strategy (1)

1 Recall from the worst-case and best-case analysis,

$$\boldsymbol{T(N) = T(i) + T(N - i - 1) + cN \quad if\ N > 1}$$

2 If the pivot is randomly selected, then each of the sizes for a partition is equally likely, thus the expected value of $\boldsymbol{T(i)}$ and $\boldsymbol{T(N - i - 1)}$ is

$$\frac{1}{N}\sum_{j=0}^{N-1} T(j)$$

then

$$T(N) = \frac{2}{N}\sum_{j=0}^{N-1} T(j) + cN$$

3 Multiplying by $\boldsymbol{N}$,

$$NT(N) = 2\sum_{j=0}^{N-1} T(j) + cN^2$$

4 Substituting $\boldsymbol{N - 1}$ for N,

$$(N-1)T(N-1) = 2\sum_{j=0}^{N-2} T(j) + c(N-1)^2$$

Average-Case Analysis with a Random Partition Strategy (2)

$$NT(N) = 2\sum_{j=0}^{N-1} T(j) + cN^2$$

$$(N-1)T(N-1) = 2\sum_{j=0}^{N-2} T(j) + c(N-1)^2$$

5 Subtracting the two,

$$\boldsymbol{NT(N) - (N-1)T(N-1) = 2T(N-1) + 2cN - c}$$

6 Rearranging and dropping the insignificant $\boldsymbol{-c}$,

$$\boldsymbol{NT(N) = (N+1)T(N-1) + 2cN}$$

7 Dividing by $\boldsymbol{N(N+1)}$,

$$\boldsymbol{\frac{T(N)}{N+1} = \frac{T(N-1)}{N} + \frac{2c}{N+1}}$$

Average-Case Analysis with a Random Partition Strategy (3)

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

7 Telescoping,

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

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

$$\frac{T(N-2)}{N-1} = \frac{T(N-3)}{N-2} + \frac{2c}{N-1}$$

$$\frac{T(2)}{3} = \frac{T(1)}{2} + \frac{2c}{3}$$

8 Collecting the terms,

$$\frac{T(N)}{N+1} = \frac{T(1)}{2} + 2c\underline{\sum_{i=3}^{N+1}\frac{1}{i}}$$

A n^{th} harmonic number,

$$H_n = \sum_{i=1}^{n}\frac{1}{i} \approx \int_1^n \frac{1}{x}dx = \ln(n)$$

8.1 Comparing the second term with the n^{th} harmonic number

$$H_n = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n-1} + \frac{1}{n}$$

$$H_{n+1} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n-1} + \frac{1}{n} + \frac{1}{n+1}$$

$$\sum_{i=3}^{N+1}\frac{1}{i} = \frac{1}{3} + \ldots + \frac{1}{N-1} + \frac{1}{N} + \frac{1}{N+1}$$

Thus,

$$2c\sum_{i=3}^{N+1}\frac{1}{i} = H_{n+1} - \frac{3}{2} \approx \ln(N+1) - \frac{3}{2}$$

Average-Case Analysis with a Random Partition Strategy (4)

$$\frac{\boldsymbol{T(N)}}{\boldsymbol{N+1}} = \frac{\boldsymbol{T(1)}}{\boldsymbol{2}} + \boldsymbol{2c}\left(\mathbf{ln}(\boldsymbol{N}+\mathbf{1}) - \frac{\mathbf{3}}{\mathbf{2}}\right)$$

B

$$\ln(N+1) = \frac{\log_2(N+1)}{\log_2 e}$$

$$= \ln 2 \lg(N+1)$$

$$\approx 0.693 \lg(N+1)$$

9 Given $\boldsymbol{T(1) = 1}$,

$$\boldsymbol{T(N)} = \frac{\boldsymbol{N+1}}{\boldsymbol{2}} + \mathbf{1.39}\boldsymbol{c}(\boldsymbol{N}+\mathbf{1})\mathbf{lg}(\boldsymbol{N}+\mathbf{1}) - \mathbf{3}\boldsymbol{c}(\boldsymbol{N}+\mathbf{1})$$

10 Ignoring the lower terms and constants

$$\boldsymbol{T(N) = O(N\mathbf{lg}N)}$$

The analysis does not apply to the partitioning strategies that do not preserve the randomness of the subarrays.

Quicksort Implementation (Optimized with Median of Three Partitioning Strategy)

```
private static final int CUTOFF = 3;

private static <E extends Comparable<E>>
void quicksort(E[] data, int left, int right) {
  if (left + CUTOFF <= right) {
    E pivot = medianOfThree(data, left, right);

    // Begin partitioning
    int i = left, j = right - 1;
    for (;;) {
      while (data[++i].compareTo(pivot) < 0) { }
      while (data[--j].compareTo(pivot) > 0) { }
      if (i < j)
        swap(data, i, j);
      else
        break;
    }

    swap(data, i, right - 1); // Restore pivot
    quicksort(data, left, i - 1); // Sort small elements
    quicksort(data, i + 1, right); // Sort large elements
  } else // Do an insertion sort on the subarray
    insertionSort(data, left, right);
}
```

```
private static <E> void swap(E[] data, int idx1, int idx2) {
  E temp = data[idx1];
  data[idx1] = data[idx2];
  data[idx2] = temp;
}
```

```
private static <E extends Comparable<E>>
void insertionSort(E[] data, int left, int right) {
  for (int p = left + 1; p <= right; p++) {
    E temp = data[p];
    int j;
    for (j = p; j > left && temp.compareTo(data[j - 1]) < 0; j--)
      data[j] = data[j - 1];

    data[j] = temp;
  }
}
```

```
private static <E extends Comparable<E>>
E medianOfThree(E[] data, int left, int right) {
  int centre = (left + right) / 2;
  if (data[centre].compareTo(data[left]) < 0)
    swap(data, left, centre);
  if (data[right].compareTo(data[left]) < 0)
    swap(data, left, right);
  if (data[right].compareTo(data[centre]) < 0)
    swap(data, centre, right);

  // Place pivot at position 'right-1'
  swap(data, centre, right - 1);
  return data[right - 1];
}
```

Summary of the Running Times

Algorithm	Worst-case	Average-case
Insertion Sort	$\Theta(n^2)$	$\Theta(n^2)$
Merge Sort	$\Theta(n \lg n)$	$\Theta(n \lg n)$
Heapsort	$O(n \lg n)$	--
Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$
Counting sort	$\Theta(k + n)$	$\Theta(k + n)$
Radix sort	$\Theta(d(k + n))$	$\Theta(d(k + n))$
Bucket sort	$\Theta(n^2)$	$\Theta(n)$