Algorithms
Chapter 7 Quicksort

Associate Professor: Ching-Chi Lin
林清池 副教授
chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

Outline

- **Description of Quicksort**
- Performance of Quicksort 分析效能
- A Randomized Version of Quicksort 隨機的演算法
- Analysis of Quicksort

Quicksort

- Worst-case running time: $\Theta(n^2)$. 最差的情形為$\theta(n^2)$
- Best practical choice: 在實作上最好的選擇
  - Expected running time: $\Theta(n \lg n)$. 平均時間為$\theta(n \lg n)$
  - Constants hidden in $\Theta(n \lg n)$ are small. $\theta(n \lg n) = c \cdot n \lg n$ 且c很微小
- Sorts in place.

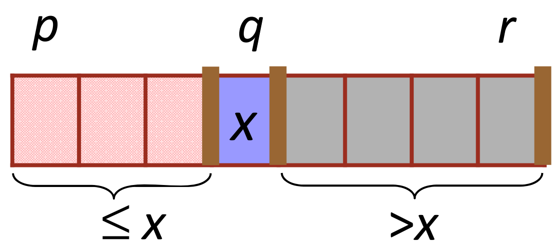
與 heap sort 一樣是 in place sorting

※雖然 heap sort 最差情形為$\theta(n \lg n) = c \cdot n \lg n$
但 c 很大，所以在實作上仍選擇 quick sort

Description of quicksort

- Quicksort is based on the three-step process of divide-and-conquer.

用 divide and conquer 解決排序



- To sort the subarray $A[p\ldots r]$:
  - **Divide:** Partition $A[p\ldots r]$, into two (possibly empty) subarrays $A[p\ldots q-1]$ and $A[q+1\ldots r]$, such that each element in the first subarray $A[p\ldots q-1]$ is $\leq A[q]$ and $A[q]$ is $<$ each element in the second subarray $A[q+1\ldots r]$.
  - **Conquer:** Sort the two subarrays by recursive calls to quicksort.
  - **Combine:** No work is needed to combine the subarrays, because they are sorted in place.

不需要 combine 的動作

The Quicksort procedure

```
QUICKSORT(A, p, r)
    if p < r                                  若左<右
        then q ← PARTITION(A, p, r)           作 divide
            QUICKSORT(A, p, q−1)              排左边 subarray
            QUICKSORT(A, q+1, r)              排右边 subarray
```

- Initial call is QUICKSORT(A, 1, n).
- Perform the divide step by a procedure PARTITION, which returns the index q.

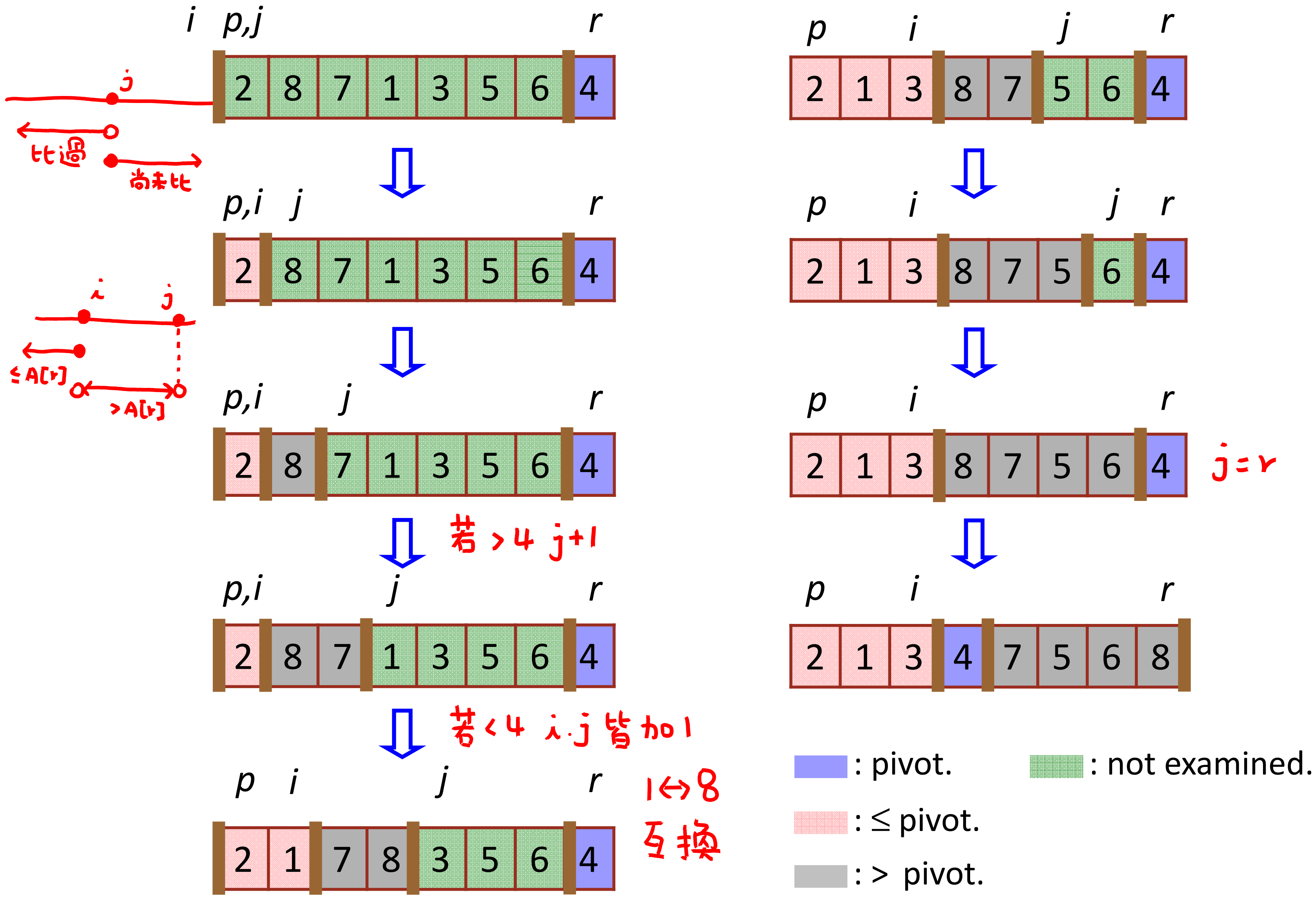
最花時間的是作 divide 的動作

Partitioning the array

- Partition subarray $A[p...r]$ by the following procedure:

```
PARTITION(A, p, r)
    x ← A[r]                          } Θ(1) 以最後一個作比較基準
    i ← p – 1                         }
    for j ← p to r – 1                }
        if A[ j ] ≤ x                 }
            i ← i + 1                 } (n–1) · Θ(1) 和x作比較
            exchange A[i ] ↔ A[j]     }
    exchange A[i+1] ↔ A[r]            } Θ(1) 將比較基準A[r]和A[i+1]互換
    return i + 1                      }      並回傳i+1
```

- PARTITION always selects the last element $A[r]$ in the subarray $A[p...r]$ as the pivot. 永遠選擇最後一個當比較基準
- Time: $\Theta(n)$.


比過
尚未比
≤A[r]
>A[r]
若>4 j+1
若<4 i.j皆加1
1↔8
互換
j=r
: pivot.
: not examined.
: ≤ pivot.
: > pivot.

Outline

- Description of Quicksort
- **Performance of Quicksort**
- A Randomized Version of Quicksort
- Analysis of Quicksort

Performance of quicksort

- The running time of quicksort depends on the partitioning of the subarrays: 時間複雜度和是否平均分割有關
  - If the subarrays are balanced, then quicksort can run asymptotically as fast as mergesort. 分割平均則和merge sort一樣快
  - If they are unbalanced, then quicksort can run asymptotically as slowly as insertion sort. 分割不平均則和insertion sort一樣差

$$T(n) = \underbrace{T(q)}_{\text{排左边}} + \underbrace{T(n-q-1)}_{\text{排右边}} + \underbrace{\theta(n)}_{\text{divide的時間}}$$

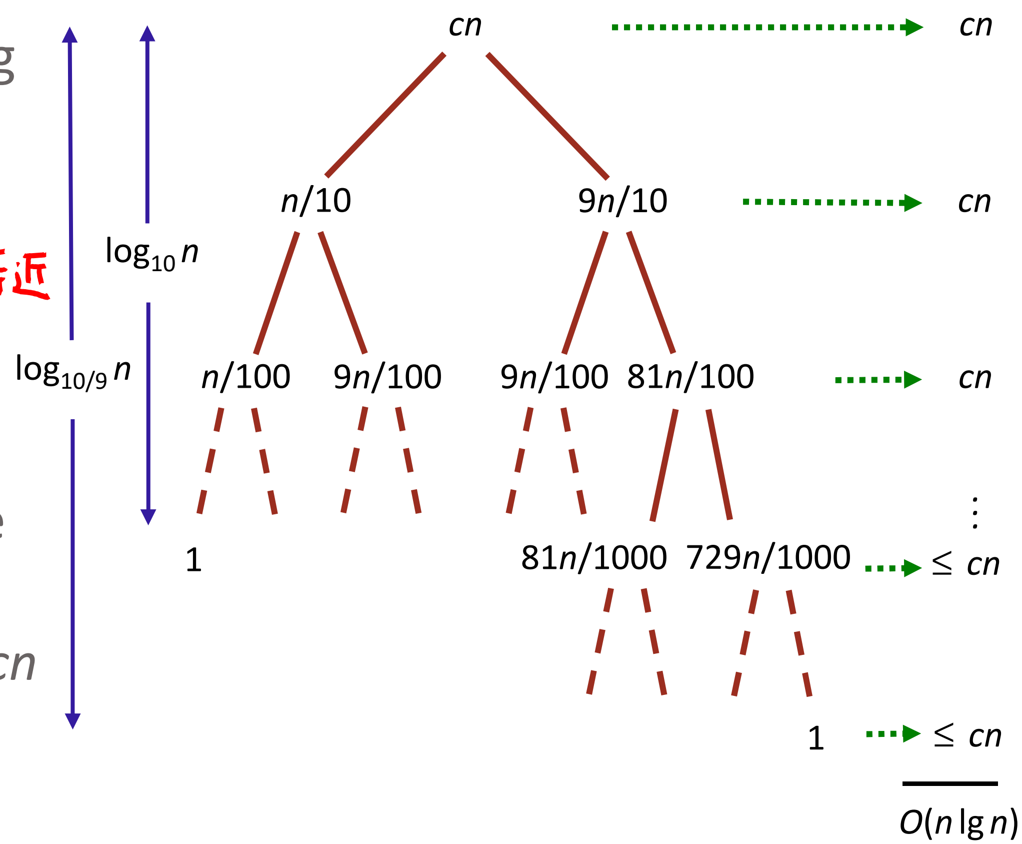
Performance of quicksort

- **Worst-case partitioning:** 分割不平均
  - Have 0 elements in one subarray and $n-1$ elements in the other subarray. 一邊0個一邊 n-1 個
  - The recurrence is $T(n) = T(n-1) + T(0) + \Theta(n)$
    $= T(n-1) + \Theta(n) = \Theta(1) + \Theta(2) + \ldots + \Theta(n)$
    $= \Theta(n^2)$.
  - Occurs when the input array is sorted.
- **Best-case partitioning:** 分割平均，剛好一半
  - Occurs when the subarrays are completely balanced every time.
  - Each subarray has $\leq n/2$ elements. 用 master method 的 case (II)
  - The recurrence is $T(n) \leq 2T(n/2) + \Theta(n)$
    $= \Theta(n \lg n)$.

Balanced partitioning$_{1/2}$

- **Balanced partitioning**
  - Quicksort's average running time is much closer to the best case than to the worst case. 平均時間和最佳時間接近
  - Imagine that PARTITION always produces a 9-to-1 split, then the running time is 假設左：右 = 9：1

    $T(n) \le T(9n/10) + T(n/10) + cn$
    $= \Theta(n \lg n)$.



用遞迴樹產生答案，用置換法驗證

Balanced partitioning$_{2/2}$

- **Intuition:** look at the recursion tree.
  - It's like the one for $T(n) = T(n/3) + T(2n/3) + O(n)$ in Section 4.2.
  - Except that here the constants are different; we get $\log_{10} n$ full levels and $\log_{10/9} n$ levels that are nonempty.

最高深度　　最低深度

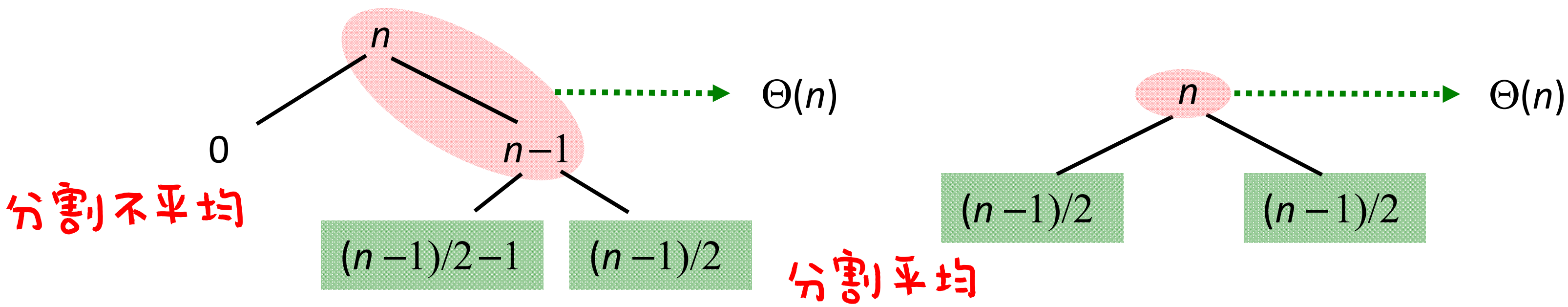
  - As long as it's a constant, the base of the log doesn't matter in asymptotic notation.
  - Any split of **constant proportionality** will yield a recursion tree of depth $\Theta(\lg n)$.

所以只要底數是常數，即左右比是常數比，時間就是$\Theta(n \lg n)$

$$h = \log_{\frac{10}{9}} n = \frac{\log_2 n}{\log_2 \frac{10}{9}} = c \cdot \lg n = \Theta(\lg n)$$

Intuition for the average case

- There will usually be a mix of good and bad splits throughout the recursion tree.
- Assume that levels alternate between best-case and worst-case splits. 假設好壞交錯發生



- The extra level in the left-hand figure only adds to the constant hidden in the Θ-notation. 左圖時間是右圖兩倍
  - Only twice as much work was done to get to that point.
  - Both figures result in $O(n \lg n)$ time.

Outline

- Description of Quicksort
- Performance of Quicksort
- **A Randomized Version of Quicksort** 隨機版本的 quick sort
- Analysis of Quicksort

Randomized version of quicksort$_{1/2}$

- In exploring the average-case behavior of quicksort, we have assumed that all input permutations are equally likely.
- This is not always true. 算平均時間時，假設所有排序發生機率相同
- We use random sampling, or picking one element at random.
- Don't always use $A[r]$ as the pivot.

從p到r隨機選一個作基準，不是永遠用A[r]

RANDOMIZED-PARTITION(A, p, r)

1. $i \leftarrow$ RANDOM(p, r) 從p到r隨機選一個作基準
2. exchange $A[r] \leftrightarrow A[i]$ A[i] A[r]交換
3. **return** PARTITION(A, p, r) 進行原本的partition

Randomized version of quicksort$_{2/2}$

- Randomly selecting the pivot element will, on average, cause the split of the input array to be reasonably well balanced.

隨機選一個作基準，平均而言
會將 input 分的平均

1. RANDOMIZED-QUICKSORT(A, p, r)
2. if $p < r$
3. then $q \leftarrow$ RANDOMIZED-PARTITION(A, p, r)
4. RANDOMIZED-QUICKSORT(A, p, $q-1$)
5. RANDOMIZED-QUICKSORT(A, $q+1$, r)

Outline

- Description of Quicksort
- Performance of Quicksort
- A Randomized Version of Quicksort
- **Analysis of Quicksort**

Analysis of quicksort

- We will analyze
  - the worst-case running time of QUICKSORT and RANDOMIZED-QUICKSORT (the same), and
  - the expected (average-case) running time of RANDOMIZED-QUICKSORT.

- **Worst-case analysis:** $T(n) = O(n^2)$.
  - Recurrence for the worst-case:

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n).$$ 有n種方法，取最差

  - Because PARTITION produces two subproblems, totaling size $n-1$, q ranges from 0 to $n-1$.

Worst-case analysis

- **Guess:** $T(n) \le cn^2$, for some c. 猜答案
  - Substituting our guess into the recurrence: 用置換法

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n)$$
$$\le \max_{0 \le q \le n-1} (cq^2 + c(n-q-1)^2) + \Theta(n)$$
$$= c \cdot \max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) + \Theta(n)$$

  - The maximum value of $(q^2+(n-q-1)^2)$ occurs – when q is either 0 or $n-1$. (second derivative)
  - This means that $\max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) \le (n-1)^2 = n^2 - 2n + 1.$
  - Therefore, $T(n) \le cn^2 - c(2n-1) + \Theta(n)$ 算出的結果

$$\le cn^2$$ 自標

$$= O(n^2)$$

choose c so that

$c(2n-1) \ge \theta(n)$ 找得出c值⇒得證

Average-case Analysis$_{1/5}$ 平均的時間

- **Average-case analysis:** $T(n) = O(n \log n)$.
  - The dominant cost of the algorithm is partitioning.
  - PARTITION is called at most n times. 最關鍵的花費是partition的時間
  - The amount of work that each call to PARTITION does is a constant plus the number of comparisons that are performed in its **for loop**. partition的時間 = O(1) + for loop中比較的時間
  - Let X = the total number of comparisons performed in all calls to PARTITION. 令x為所有for loop中比較的時間
  - Therefore, the **total work is $O(n + X)$**.

n · (O(1) + for loop中比較的時間) = O(n + x)

任兩個element最多比較一次
(I) 只跟基準比
(II) 基準不再跟別人比
(III) 左右往後不會再相比

Average-case Analysis$_{2/5}$

- We will now compute a bound on the overall number of comparisons.
- For ease of analysis: 方便分析
  - Rename the elements of A as $z_1, z_2, \ldots, z_n$, with z_i being the ith smallest element. z_i是第i小的元素
  - Define the set $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$ to be the set of elements between z_i and z_j, inclusive.
- Each pair of elements is compared at most once:
  - Elements are compared only to the pivot element, and
  - The pivot element is never in any later call to PARTITION.
- The expectation of total number of comparisons performed by the algorithm is $\mathrm{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$.

比較次數的期望值

Average-case Analysis$_{3/5}$

- Consider the input: 2, 8, 7, 1, 3, 5, 6, 4 and the pivot is 4,
  - None of the set {2, 1, 3} will ever be compared to any of the set {5, 6, 7,8}. 比基準大者和比基準小者，往後不會再相比
- Once a pivot x is chosen such that $z_i < x < z_j$, then z_i and z_j will never be compared at any later time.

$z_i < x < z_j$ 且 x 比 z_i 和 z_j 早選中 ⇒ z_i 和 z_j 不會互比

- If either z_i or z_j is chosen before any other element of Z_{ij}, then it will be compared to all the elements of Z_{ij}, except itself.

若 z_i or z_j 比 z_{ij} 中其他点早選中，則 z_i or z_j 將和 z_{ij} 中所有点相比

3和7比的机率
⇒ 1 和 2 先選不会影響
3 和 7 仍然同時在 1 或 2 的右边
⇒ 8 也不会同時在 8 的左边
4、5 和 6 会將 3 和 7 分開

⇒ 3 或 7 要比 4、5 和 6 先選
⇒ 3 先選机率 $\frac{1}{7-3+1}$
⇒ 7 先選机率 $\frac{1}{7-3+1}$
⇒ 7 和 3 比的机率 $\frac{2}{7-3+1}$

Average-case Analysis$_{4/5}$

- Therefore,

Pr{z_i is compared to z_j} = Pr{z_i or z_j is the first pivot chosen from Z_{ij}}
= Pr{z_i is the first pivot chosen from Z_{ij}}
+ Pr{z_j is the first pivot chosen from Z_{ij}}

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1}$$

z_i被選中的機率 z_j被選中的機率

$$= \frac{2}{j-i+1}.$$

- Substituting into the equation for E[X]:
  - $E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}.$ 期望值

Average-case Analysis$_{5/5}$

- Let $k = j - i$, then
$$
\begin{aligned}
\mathrm{E}[X] &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1}\sum_{k=1}^{n-i}\frac{2}{k+1} \\
&< \sum_{i=1}^{n-1}\sum_{k=1}^{n}\frac{2}{k} \\
&= \sum_{i=1}^{n-1}O(\lg n) \\
&= O(n\lg n).
\end{aligned}
$$

$k = j - i$

$$
\begin{aligned}
&\sum \frac{2}{k} \\
&= 2\sum\frac{1}{k} \\
&= 2\left(\frac{1}{1}+\frac{1}{2}+\dots+\frac{1}{n}\right) \\
&= 2(\lg n + O(1)) \\
&= O(\lg n)
\end{aligned}
$$

- So the expected running time of quicksort, using RANDOMIZED-PARTITION, is $O(n\lg n)$.