Quicksort

Algorithm : Design & Analysis

[4]

In the last class…

- Recursive Procedures
- Proving Correctness of Recursive Procedures
- Deriving recurrence equations
- Solution of the Recurrence equations
  - Guess and proving
  - Recursion tree
  - Master theorem
- Divide-and-conquer

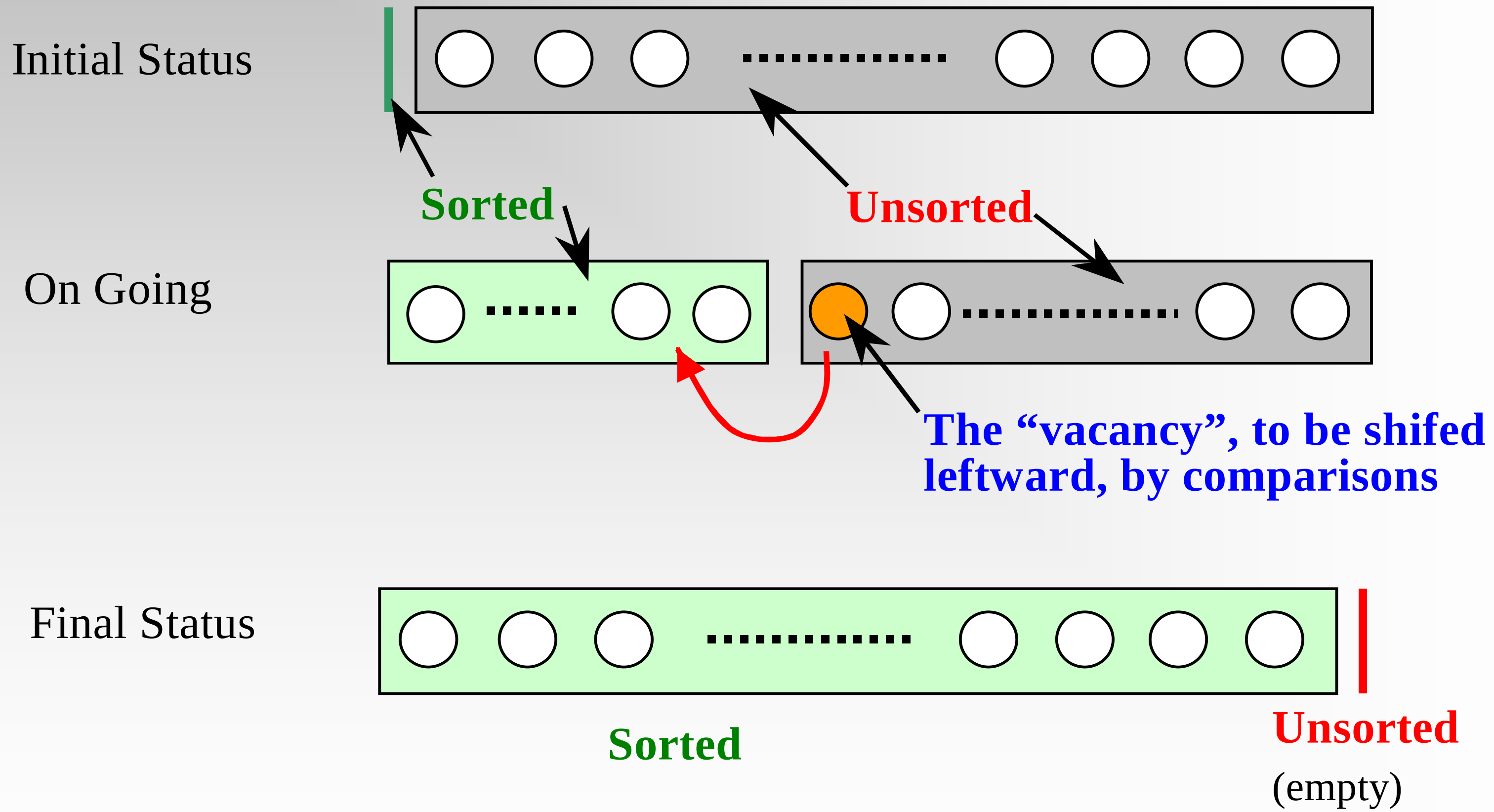
Quicksort

- Insertion sort
- Analysis of insertion sorting algorithm
- Lower bound of local comparison based sorting algorithm
- General pattern of divide-and-conquer
- Quicksort
- Analysis of Quicksort

Comparison-Based Algorithm

- The class of “algorithms that sort by comparison of keys”
  - comparing (and, perhaps, copying) the key
  - no other operations are allowed
- The measure of work used for analysis is the number of comparison.

As Simple as Inserting



Shifting Vacancy: the Specification

- **int** shiftVac(Element[ ] E, **int** vacant, Key x)
- *Precondition*: vacant is nonnegative
- *Postconditions*: Let xLoc be the value returned to the caller, then:
  - Elements in *E* at indexes less than xLoc are in their original positions and have keys less than or equal to *x*.
  - Elements in *E* at positions (xLoc+1,…, vacant) are greater than *x* and were shifted up by one position from their positions when shiftVac was invoked.

Shifting Vacancy: Recursion

```
int shiftVacRec(Element[] E, int vacant, Key x)
    int xLoc
if (vacant==0)
    xLoc=vacant;
else if (E[vacant-1].key≤x)
    xLoc=vacant;
else
    E[vacant]=E[vacant-1];
    xLoc=shiftVacRec(E,vacant-1,x);
Return xLoc
```

The recursive call is working on a smaller range, so terminating;

The second argument is non-negative, so precondition holding

Worse case frame stack size is $O(n)$

Shifting Vacancy: Iteration

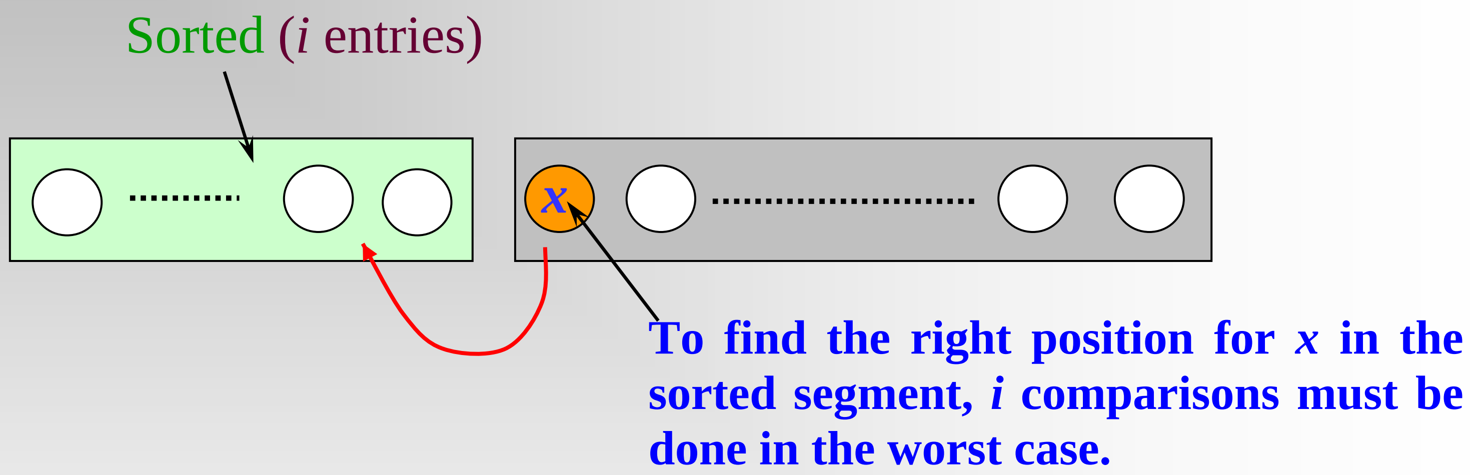
```
int shiftVac(Element[] E, int xindex, Key x)
    int vacant, xLoc;
    vacant=xindex;
    xLoc=0;  //Assume failure
    while (vacant>0)
        if (E[vacant-1].key≤x)
            xLoc=vacant;   //Succeed
            break;
        E[vacant]=E[vacant-1];
        vacant--;   //Keep Looking
    return xLoc
```

Insertion Sorting: the Algorithm

- Input: *E*(array), $n \geq 0$(size of *E*)
- Output: *E*, ordered nondecreasingly by keys
- Procedure:

```
void insertSort(Element[] E, int n)
    int xindex;
    for (xindex=1; xindex<n; xindex++)
        Element current=E[xindex];
        Key x=current.key;
        int xLoc=shiftVac(E,xindex,x);
        E[xLoc]=current;
    return;
```

Worst-Case Analysis

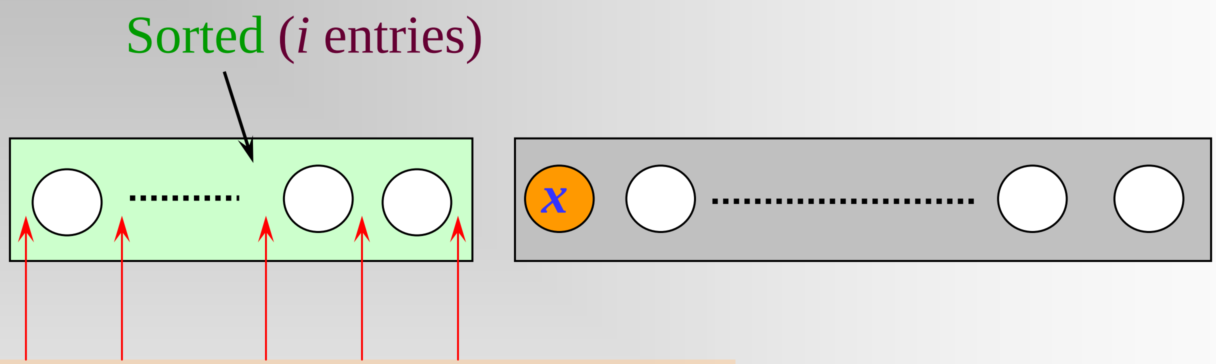


- At the beginning, there are *n*-1 entries in the unsorted segment, so:

$$W(n) \leq \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

The input for which the upper bound is reached does exist, so: $W(n) \in \Theta(n^2)$

Average Behavior



***x* may be located in any one of the *i*+1 intervals(inclusive), assumingly, with the same probabiliy**

- Assumptions:
  - All permutations of the keys are equally likely as input.
  - There are not different entries with the same keys.

Note: For the (i+1)th interval (leftmost), only i comparisons are needed.

Average Complexity

- The average number of comparisons in **shiftVac** to find the location for the *i*th element:

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

for the leftmost interval

- For all *n*-1 insertions:

$$A(n) = \sum_{i=1}^{n-1}\left(\frac{i}{2} + 1 - \frac{1}{i+1}\right) = \frac{n(n-1)}{4} + n - 1 - \sum_{j=2}^{n}\frac{1}{j}$$

$$= \frac{n(n-1)}{4} + n - \sum_{j=1}^{n}\frac{1}{j} = \frac{n^2}{4} + \frac{3n}{4} + \ln n \in \Theta(n^2)$$

Inversion and Sorting

- An unsorted sequence *E*:

  $$x_1, x_2, x_3, \ldots, x_{n-1}, x_n$$

- If there are no same keys, for the purpose of sorting, it is a reasonable assumption that $\{x_1, x_2, x_3, \ldots, x_{n-1}, x_n\}=\{1,2,3,\ldots,n-1,n\}$
- $\langle x_i, x_j \rangle$ is an ***inversion*** if $x_i>x_j$, but $i<j$
- All the inversions ***must*** be eliminated during the process of sorting

Eliminating Inverses: Worst Case

- Local comparison is done between two adjacent elements.
- At most ***one*** inversion is removed by a local comparison.
- There do exist inputs with ***n*(*n*-1)/2** inversions, such as (n,n-1,…,3,2,1)

- **The worst-case behavior of any sorting algorithm that remove at most one inversion per key comparison must in $\Omega(n^2)$**

Elininating Inverses: Average

- Computing the average number of inversions in inputs of size n (n>1):
  - Transpose:

    $x_1, x_2, x_3, \ldots, x_{n-1}, x_n$

    $x_n, x_{n-1}, \ldots, x_3, x_2, x_1$

  - For any i, j, ($1 \le j \le i \le n$), the inversion (x_i, x_j) is in exactly one sequence in a transpose pair.
  - The number of inversions (x_i, x_j) on n distinct integers is $n(n-1)/2$.
  - So, the average number of inversions in all possible inputs is $\boldsymbol{n(n-1)/4}$**,** since exactly $n(n-1)/2$ inversions appear in each transpose pair.
- **The average behavior of any sorting algorithm that remove at most one inversion per key comparison must in $\Omega(n^2)$**

Traveling a Long Way

- Problem
  - If $a_1, a_2, \ldots a_n$ is a random permutation of $\{1,2,\ldots n\}$, what is the average value of

    $| a_1$

- The answer is the averag
  during a sorting process.

For a spcific $j(1 \le j \le n)$, *the*

$$\sum_{j=1}^{n}\frac{1}{n}\left(\sum_{i=1}^{j-1} i+\sum_{i=1}^{n-j} i\right)=\frac{2}{n}\sum_{j=1}^{n}(1+2+\ldots+(j-1))$$

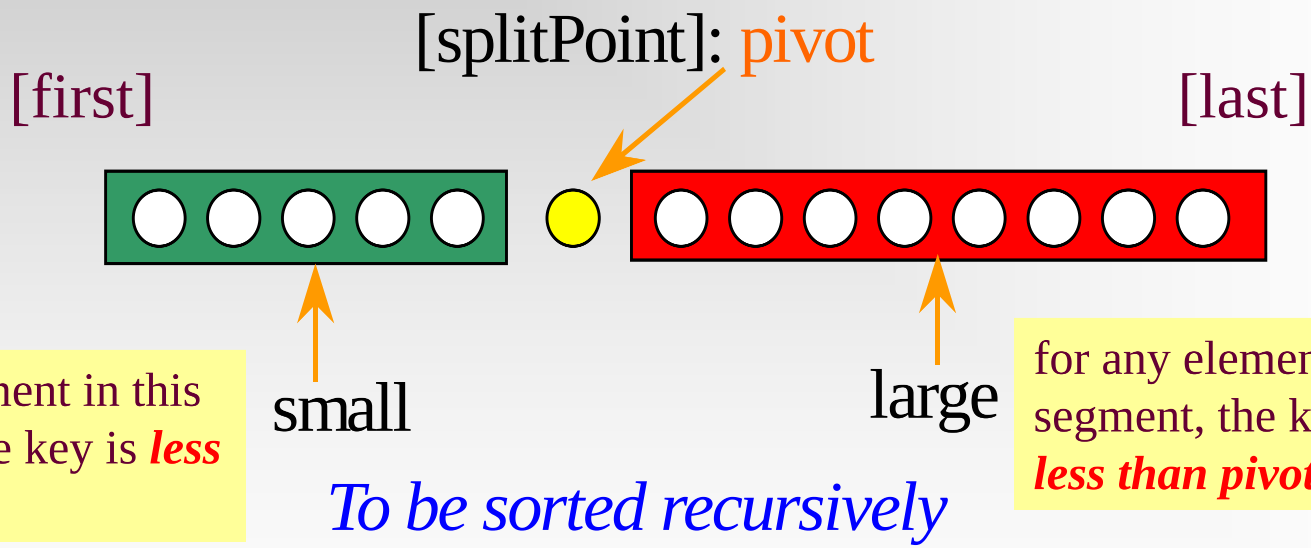
$$=\frac{1}{n}\sum_{j=1}^{n}(j^2-j)=\frac{1}{6}(n+1)(2n+1)-\frac{1}{2}(n+1)$$

$$\frac{1}{n}(|1-j|+|2-j|+\ldots+|n-j|)=\frac{1}{n}\left(\sum_{i=1}^{j-1}(j-i)+\sum_{i=j+1}^{n}(i-j)\right)=\frac{1}{n}\left(\sum_{i=1}^{j-1} i+\sum_{i=1}^{n-j} i\right)$$

sum on j gives $\frac{1}{3}(n^2-1)$

Quicksort: the Strategy

- Dividing the array to be sorted into two parts: “small” and “large”, which will be sorted recursively.



for any element in this segment, the key is ***less than pivot***.

for any element in this segment, the key is ***not less than pivot***.

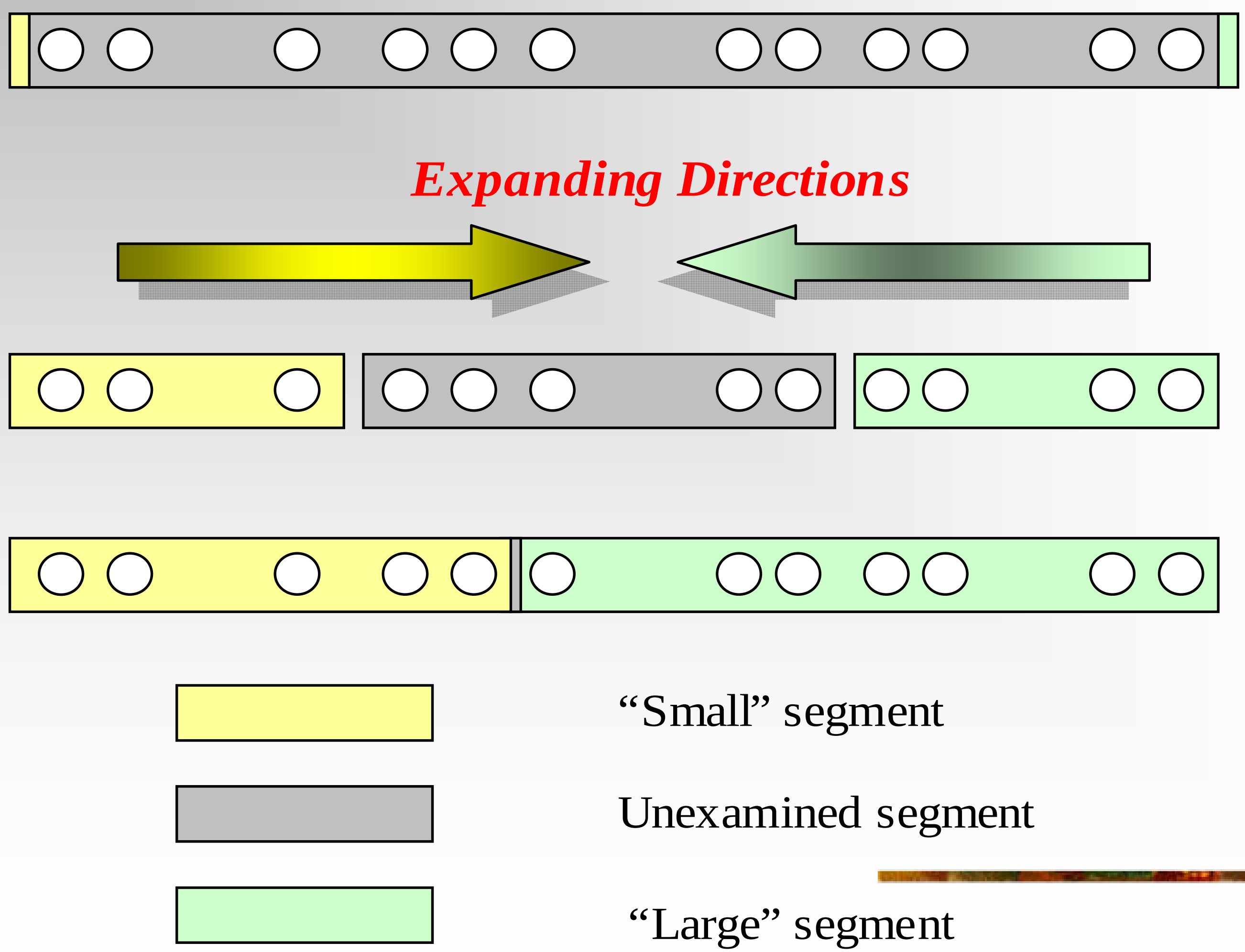
QuickSort: the algorithm

- Input: Array *E* and indexes *first*, and *last*, such that elements *E*[*i*] are defined for *first*≤*i*≤*last*.
- Output: *E*[*first*],…,*E*[*last*] is a sorted rearrangement of the same elements.
- The procedure:

```
void quickSort(Element[ ] E, int first, int last)
    if (first<last)
        Element pivotElement=E[first];
        Key pivot=pivotElement.key;
        int splitPoint=partition(E, pivot, first, last);
        E[splitPoint]=pivotElement;
        quickSort(E, first, splitPoint-1);
        quickSort(E, splitPoint+1, last);
    return
```

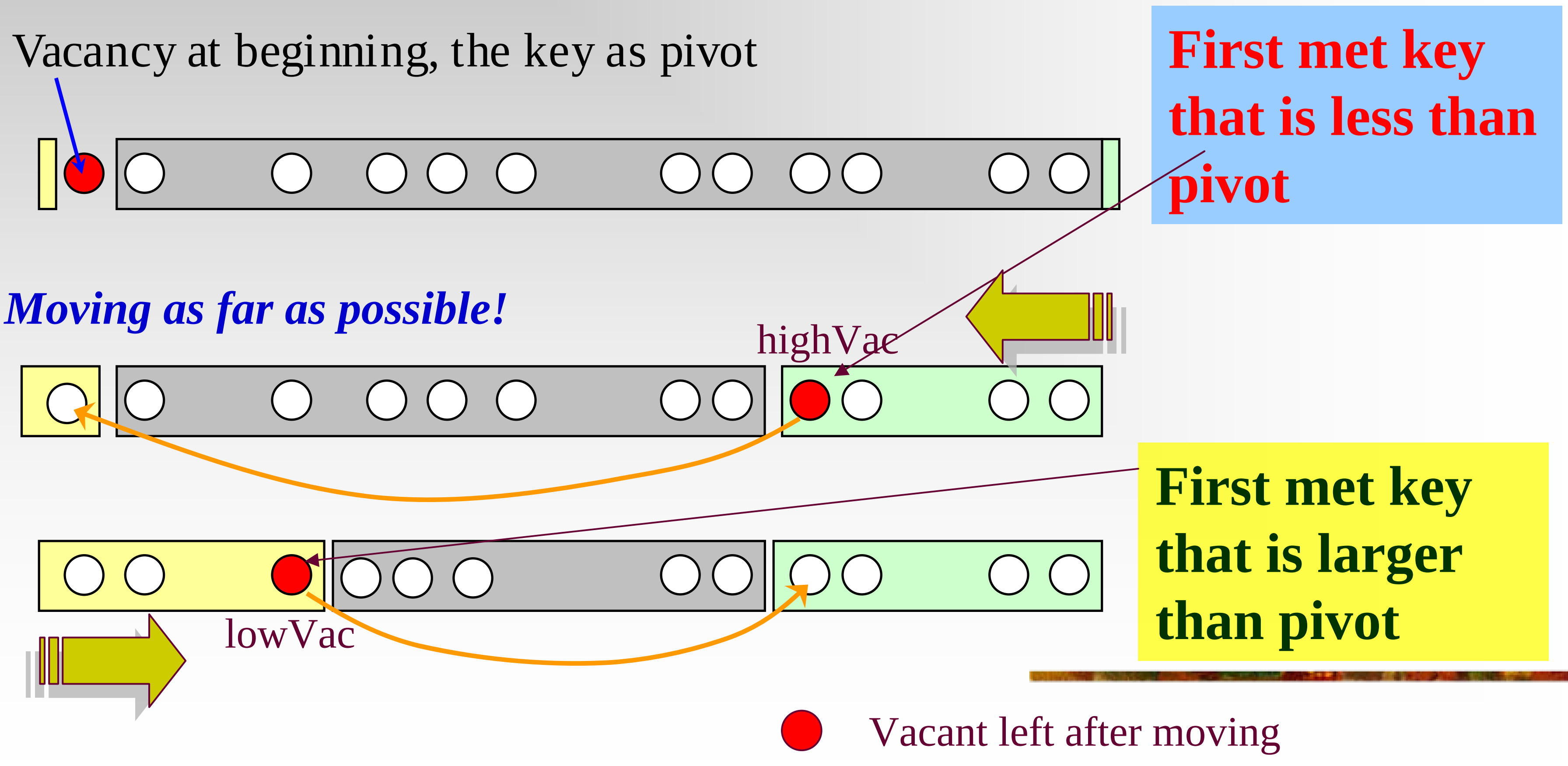
The splitting point is chosen arbitrarily, as the first element in the array segment here.

Partition: the Strategy



Partition: the Process

- Always keep a vacancy before completion.



Partition: the Algorithm

- Input: Array E, pivot, the key around which to partition, and indexes *first*, and *last*, such that elements $E[i]$ are defined for $first+1 \le i \le last$ and $E[first]$ is vacant. It is assumed that $first < last$.
- Output: Returning *splitPoint*, the elements origingally in $first+1, \ldots, last$ are rearranged into two subranges, such that
  - the keys of $E[first], \ldots, E[splitPoint-1]$ are less than pivot, and
  - the keys of $E[splitPoint+1], \ldots, E[last]$ are not less than pivot, and
  - $first \le splitPoint \le last$, and $E[splitPoint]$ is vacant.

Partition: the Procedure

int partition(Element [ ] E, Key pivot, **int** first, **int** last)
    int low, high;
1. low=first; high=last;
2. **while** (low<high)
3.   **int** highVac=extendLargeRegion(E,pivot,low,high);
4.   **int** lowVac =
        extendSmallRegion(E,pivot,low+1,highVac);
5.   low=lowVac; high=highVac-1;
6 **return** low; *//This is the splitPoint*

highVac has been filled now.

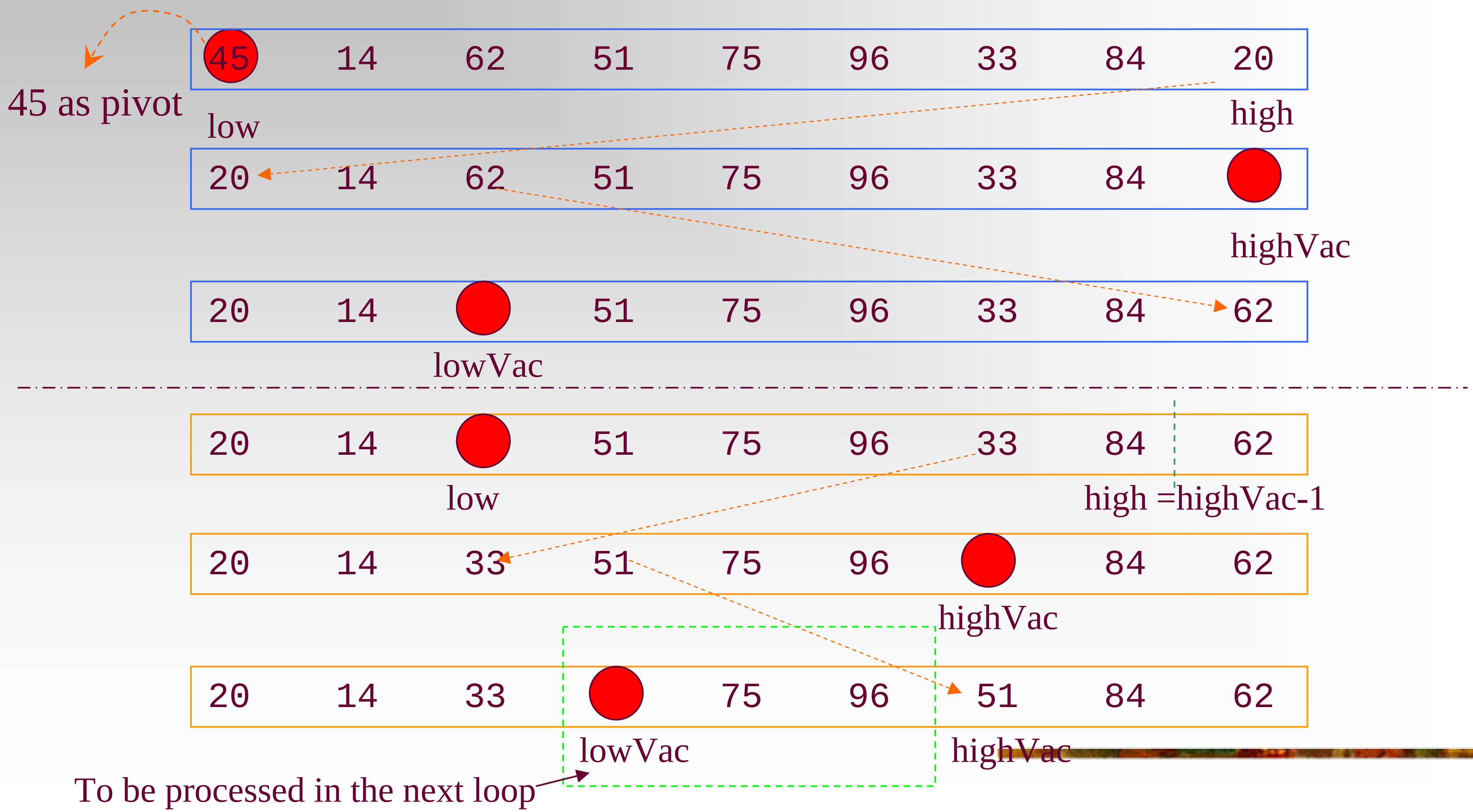
Extending Regions

- Specification for

extendLargeRegion(Element[ ] E, Key pivot, **int** lowVac, **int** high)

  - Precondition:
    - lowVac<high
  - Postcondition:
    - If there are elements in *E[lowVac+1],...,E[high]* whose key is less than pivot, then the rightmost of them is moved to *E[lowVac]*, and its original index is returned.
    - If there is no such element, *lowVac* is returned.

Example of Quicksort



Divide and Conquer: General Pattern

solve(I)
  n=size(I);
  if ($n \le$smallSize)
    solution=directlySolve(I)
  else
    divide I into $I_1, \ldots I_k$;
    for each $i \in \{1,\ldots,k\}$
      S_i=solve(I_i);
    solution=combine($S_1 ,\ldots ,S_k$);
  return solution

$T(n)=B(n)$ for $n \le$smallSize

$$T(n)=D(n)+\sum_{i=1}^{k} T(size(I_i))+C(n)$$

for $n>$smallSize

Workhorse

- “Hard division, easy combination”
- “Easy division, hard combination”

- Usually, the “real work” is in one part.

Worst Case: a Paradox

- For a range of *k* positions, *k*-1 keys are compared with the pivot(one is vacant).
- If the pivot is the smallest, than the "large" segment has all the remaining *k*-1 elements, and the "small" segment is empty.
- If the elements in the array to be sorted has already in ascending order(the ***Goal***), then the number of comparison that Partition has to do is:

$$\sum_{k=2}^{n}(k-1)=\frac{n(n-1)}{2}\in O(n^2)$$

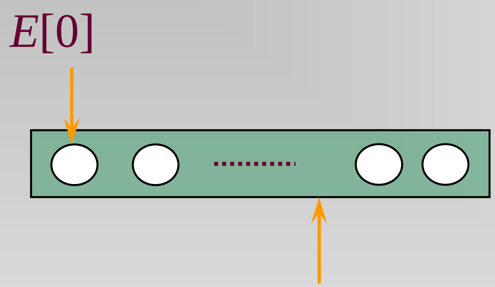
Average Analysis

- Assumption: all permutation of the keys are ***equally likely***.
- A(n) is the average number of key comparison done for range of size n.
- In the first cycle of *Partition*, n-1 comparisons are done
- If split point is $E[i]$(each i has probability $1/n$), *Partition* is to be executed recursively on the subrange $[0,\ldots,i]$ and $[i+1,\ldots,n-1]$

The Recurrence E

Why the assumed probability is still hold for each subrange?

No two keys within a subrange have been compared each other!



subrange 1: size= ***i*** subrange 2: size= **n-1-*i***

with $i \in \{0,1,2,\ldots n\text{-}1\}$, each value with the probability **1/*n***

So, the average number of key comparison $A(n)$ is:

$$A(n) = (n-1) + \sum_{i=0}^{n-1} \frac{1}{n}[A(i) + A(n-1-i)] \quad \mathit{for}\ n \geq 2$$

and $A(1)=A(0)=0$

The number of key comparison in the first cycle(finding the splitPoint) is n-1

Simplified Recurrence Equation

- Note: $\sum_{i=0}^{n-1} A(i) = \sum_{i=0}^{n-1} A[(n-1)-i] \quad and \quad A(0) = 0$

- So: $A(n) = (n-1) + \frac{2}{n}\sum_{i=1}^{n-1} A(i) \quad for\, n \geq 1$

- Two approaches to solve the equation
  - Guess and prove by induction
  - Solve directly

Guess the Solution

- A special case as clue for guess
  - Assuming that *Partition* divide the problem range into 2 subranges of about the same size.
  - So, the number of comparison $Q(n)$ satisfy:

$$Q(n) \approx n+2Q(n/2)$$

  - Applying *Master Theorem*, case 2:

$$\boldsymbol{Q(n)\in\Theta(n\log n)}$$

Note: here, $b=c=2$, so $E=\lg(b)/\lg(c)=1$, and, $f(n)=n=n^E$

Inductive Proof: A(n)∈O($n$ln$n$)

- Theorem: **$A(n) \le cn\ln n$** for some constant c, with $A(n)$ defined by the recurrence equation above.
- Proof:
  - By induction on n, the number of elements to be sorted. Base case(n=1) is trivial.
  - Inductive assumption: **$A(i) \le ci\ln i$** for $1 \le i < n$

$$A(n) = (n-1) + \frac{2}{n}\sum_{i=1}^{n-1} A(i) \le (n-1) + \frac{2}{n}\sum_{i=1}^{n-1} ci\ln(i)$$

$$Note\ : \frac{2}{n}\sum_{i=1}^{n-1} ci\ln(i) \le \frac{2c}{n}\int_1^n x\ln x dx \approx \frac{2c}{n}\left(\frac{n^2\ln(n)}{2} - \frac{n^2}{4}\right) = cn\ln(n) - \frac{cn}{2}$$

$$So, \quad A(n) \le cn\ln(n) + n\left(1 - \frac{c}{2}\right) - 1$$

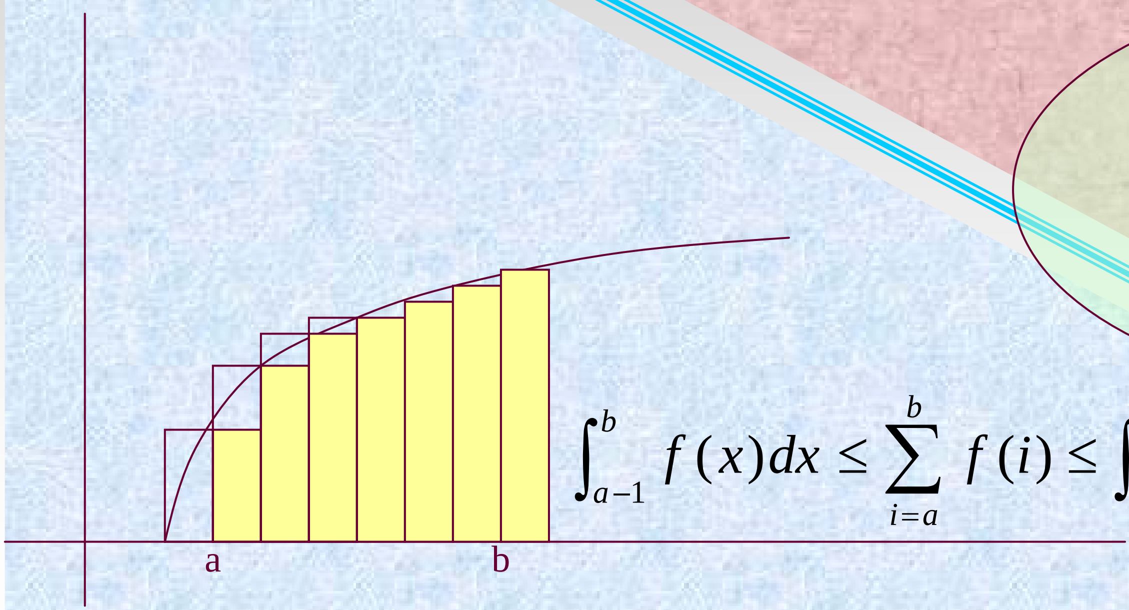
$$Let\ c = 2,\ we\ have\ \ A(n) \le 2n\ln(n)$$

For Your Reference

$$\int_1^n x^k \ln(x)dx = \frac{1}{k+1}n^{k+1}\ln(n) - \frac{1}{(k+1)^2}n^{k+1}$$

$$\sum_{i=1}^{n} \frac{1}{i} \approx \ln(n) + 0.577$$

Harmonic Series



$$\int_{a-1}^{b} f(x)dx \le \sum_{i=a}^{b} f(i) \le \int_{a}^{b+1} f(x)dx$$

Inductive Proof: A(n)$\in\Omega$($n$ln$n$)

- Theorem: **$A(n)>cn\ln n$** for some co c, with **large n**
- Inductive reasoning:

Inductive assumption

$$A(n)=(n-1)+\frac{2}{n}\sum_{i=1}^{n-1}A(i)>(n-1)+\frac{2}{n}\sum_{i=1}^{n-1}ci\ln(i)$$

$$=(n-1)+\frac{2c}{n}\sum_{i=2}^{n}i\ln(i)-2c\ln(n)\geq(n-1)+\frac{2c}{n}\int_{1}^{n}x\ln x dx-2c\ln(n)$$

$$\approx cn\ln(n)+[(n-1)-c(\frac{n}{2}+2\ln n)]$$

$$Let\, c<\frac{n-1}{\frac{n}{2}+2\ln(n)},\quad then\quad A(n)>cn\ln(n)\ \ (Note: \lim_{n\to\infty}\frac{n-1}{\frac{n}{2}+2\ln(n)}=2)$$

Directly Derived Recurrence Equation

$$We\ have: \quad A(n) = (n-1) + \frac{2}{n}\sum_{i=1}^{n-1} A(i) \quad and$$

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

Combining the 2 equations in some way, we can remove all A(i) for i=1,2,…,n-2

$$nA(n) - (n-1)A(n-1)$$

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

$$= 2A(n-1) + 2(n-1)$$

$$So,\ nA(n) = (n+1)A(n-1) + 2(n-1)$$

Solving the Equation

Let it be $B(n)$

$$nA(n) = (n+1)A(n-1) + 2(n-1) \Rightarrow \boxed{\frac{A(n)}{n+1}} = \frac{A(n-1)}{n} + \frac{2(n-1)}{n(n+1)}$$

We have equation: $B(n) = B(n-1) + \dfrac{2(n-1)}{n(n+1)}$ $\quad B(1) = 0$

Note $\dfrac{1}{i(i+1)} = \dfrac{1}{i} - \dfrac{1}{i+1}$

$$B(n) = \sum_{i=1}^{n} \frac{2(i-1)}{i(i+1)} = 2\sum_{i=1}^{n} \frac{(i+1)-2}{i(i+1)} = 2\sum_{i=1}^{n} \frac{1}{i} - 4\sum_{i=1}^{n} \frac{1}{i(i+1)}$$

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

So, $\quad B(n) \approx 2(\ln n + 0.577) - \dfrac{4n}{n+1}, \quad$ *and,* $A(n) \approx 1.386 n \lg n - 2.846 n$

Note: $\ln n \approx 0.693 \lg n$

Space Complexity

- Good news:
  - Partition is in-place
- Bad news:
  - In the worst case, the depth of recursion will be n-1
  - So, the largest size of the recursion stack will be in $\Theta(n)$

Home Assignment

- pp.208-
  - 4.6
  - 4.8-4.9
  - 4.11-4.12
  - 4.17-4.18
  - 4.21-4.22