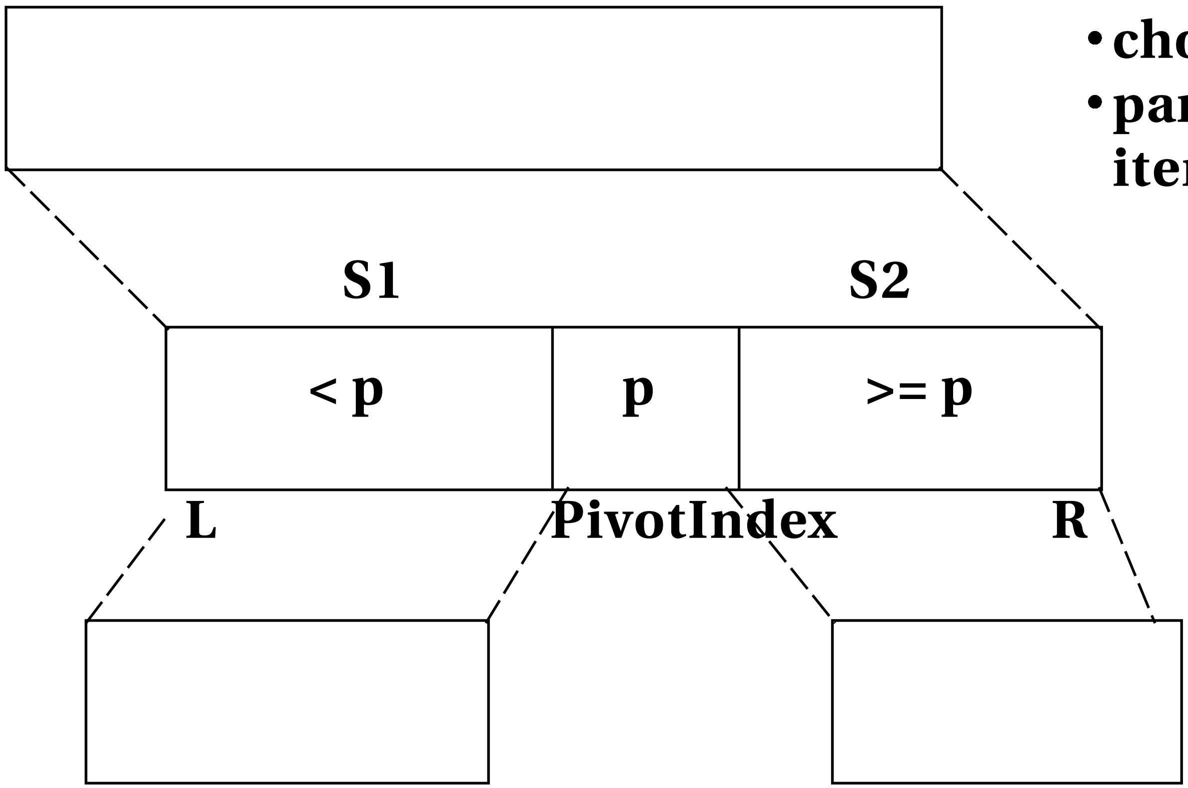
Quick Sort (overview)

- Divide and Conquer Algorithm



- choose pivot
- partition the items

\pn

Quick Sort (pivot)

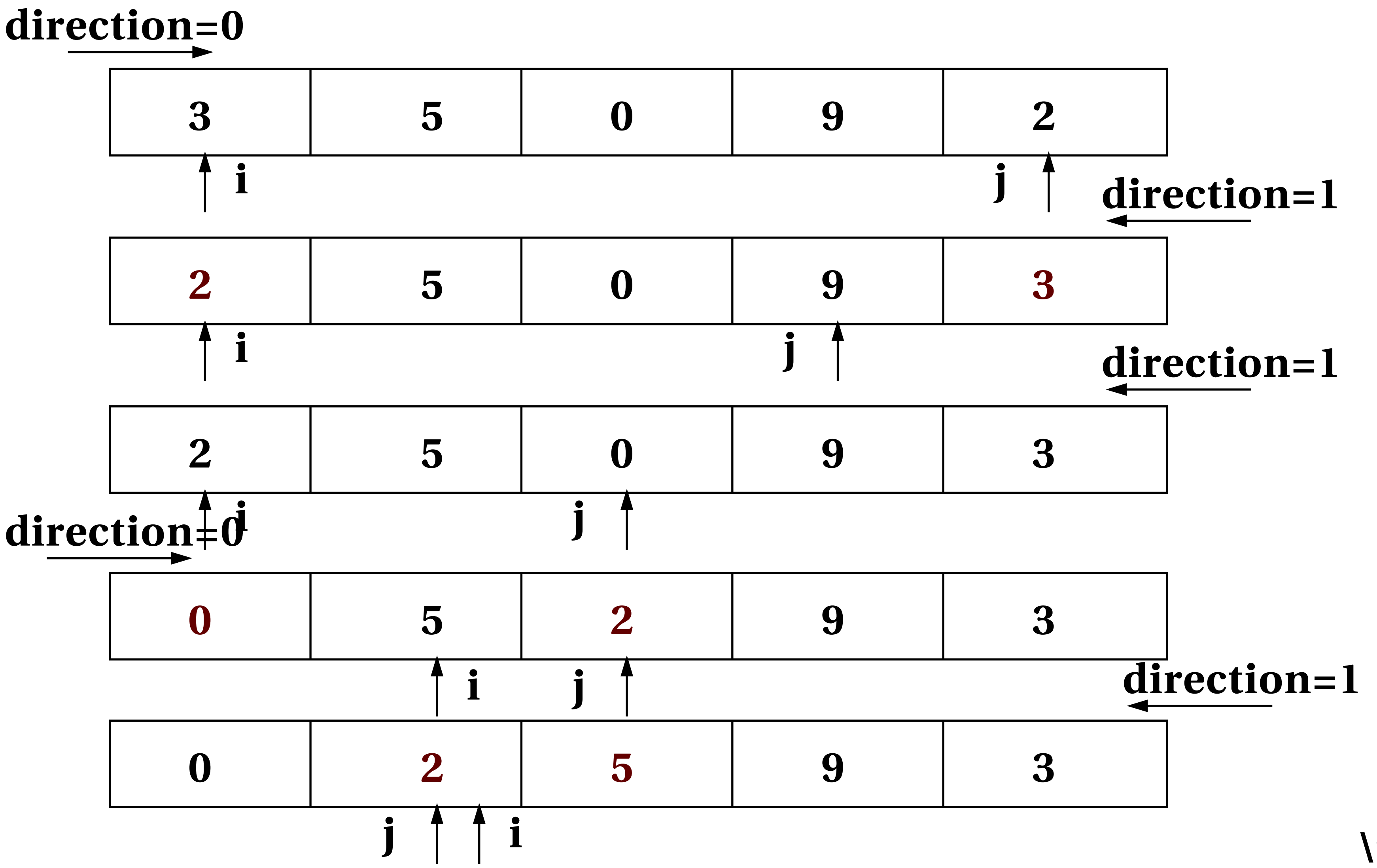
- Methods (book):
  - First element
  - Random element
  - *Median of the array*

p	< p	>= p	Unknown

OR

\pn

Quick Sort (partition)



\pn

Quick Sort (partition 2)

```
void Partition (int A[], int L, int F, int &PivotIndex)
{
 int direction = 0, i = L, j = F;

 while (i < j) {
  if (A[i] > A[j]) {
   direction = 1 - direction;
   Swap (A[i], A[j]);
  }
  if (direction)
   j--;
  else
   i++;
 }
 PivotIndex = i;
}
```

Quick Sort (algorithm)

```
void QuickSrt (int A[], int L, int R)
{
 int PivotIndex = 0;

 if (R > L)
  {
   Partition (A, L, R, PivotIndex);
   QuickSrt (A, L, PivotIndex - 1);
   QuickSrt (A, PivotIndex + 1, R);
  }
}

void QuickSortR (int A[], int N) { QuickSrt (A, 0, N-1); }
```

Quick Sort (iter.)

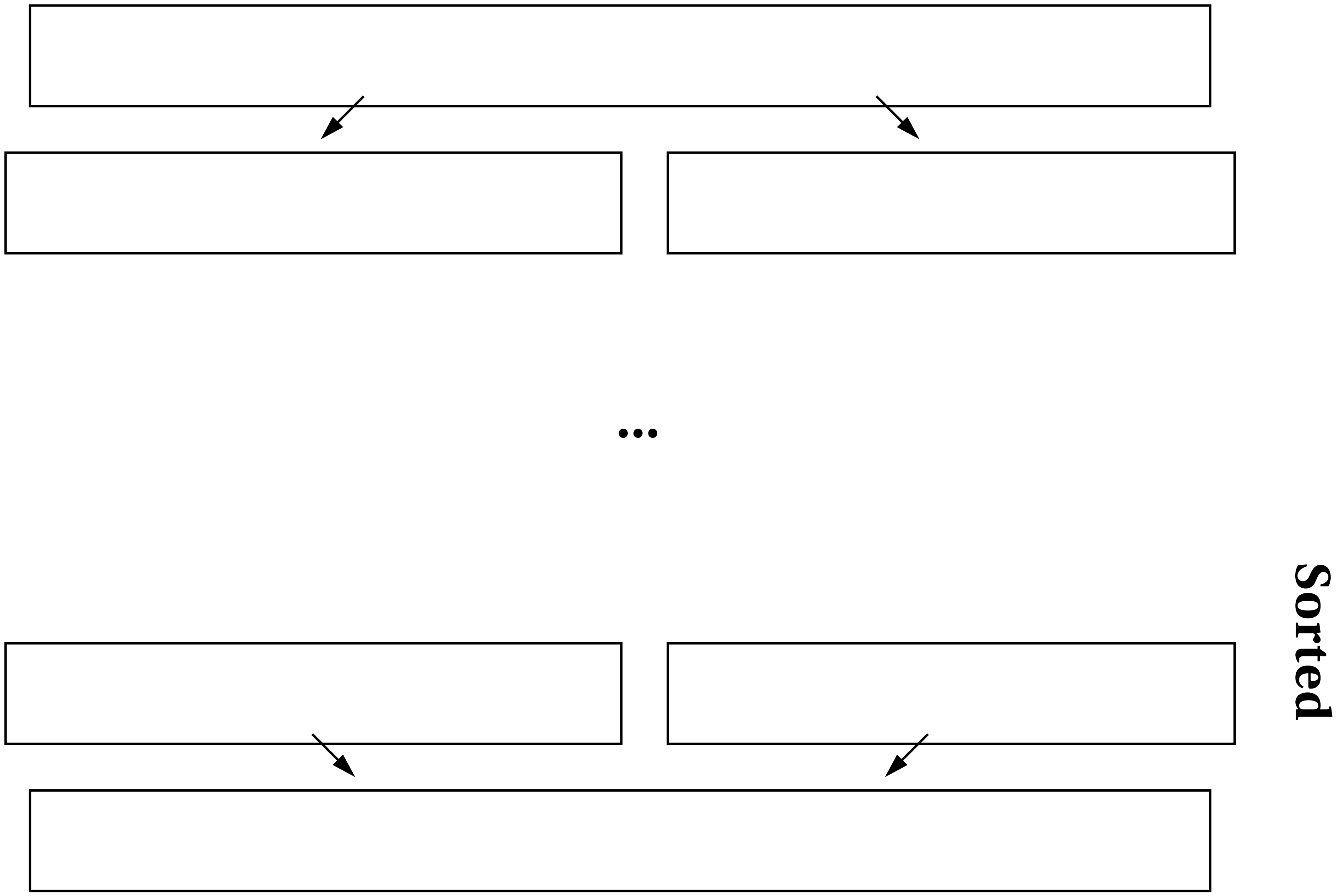
```
void QuickSortI (int A[], int N)
{
 int PivotIndex, L, R;
 Stack<int> S;

 S.Push (0); S.Push (N-1);
 while (!S.isEmpty ()) {
  S.Pop (R); S.Pop (L);

  if (R <= L) continue;

  Partition (A, L, R, PivotIndex);
  S.Push (L); S.Push (PivotIndex-1);// QuickSrt (A, L, Pivot-1)
  S.Push (PivotIndex+1); S.Push (R);// QuickSrt (A, Pivot+1, R)

 }
}
```

\pn

Quick Sort (times)

- Worst Case: O(N^2)
- Average Case: O(N* log (N))
- Best Case: O(N*log(N))

Merge Sort (overview)



\pn

Merge Sort (rec.)

```
void MSort (int A[], int TmpA[], int Left, int Right)
{
 int Center;
 if (Left < Right) {
  Center = (Left + Right) / 2;
  MSort (A, TmpA, Left, Center);
  MSort (A, TmpA, Center+1, Right);
  Merge (A, TmpA, Left, Center+1, Right);
 }
}

void MergeSort (int A[], int N)
{
 int TmpA[N];
 MSort (A, TmpA, 0, N-1);
}
```

Merge (2 sorted arrays)

```
void Merge (int A[], int TmpA[], int LPos, int MPos, int RPos)
{
 int LEnd = MPos - 1, N = RPos - LPos + 1, TmpPos = LPos;

 while ((LPos <= LEnd) && (MPos <= RPos)) {
  if (A[LPos] <= A[MPos])
   TmpA[TmpPos++] = A[LPos++];
  else
   TmpA[TmpPos++] = A[MPos++];
 }
 while (LPos <= LEnd)
  TmpA[TmpPos++] = A[LPos++];
 while (MPos <= RPos)
  TmpA[TmpPos++] = A[MPos++];

 for (int i = 0; i < N; i++, RPos--)
  A[RPos] = TmpA[RPos];
}
```

\pn

Merge Sort (iter.)

```
void MergeSortI (int A[], int N)
{
 for (int m = 1; m < N; m += m)
  for (int i = 1; i <= r - m; i += m+m)
   merge (a, i, i+m, min (i+m+m, r));
}
```

Merge Sort (linked list)

```
Node* MergeSortL (Node* list)
{
 Queue Q; Node *tmpRef;
 for ( ; list != NULL; list = tmpRef) {
  tmpRef = list->next;
  list->next = NULL;
  Q.Enqueue (list);
 }
 while (Q.Length () > 1) {
   list  = merge (Q.Dequeue (), Q.Dequeue ());
   Q.Enqueue (list);
 }
 return  Q.Dequeue ();
}
```

Merge Sort (times)

- Worst Case: O(N * log (N))
- Average Case: O(N * log(N))

Running Times (experiments)

	1000	1000*10	1000*100
Selection	0.00980702	1.13673	?????????
Bubble	0.031423	3.29603	?????????
Insert	0.030031	0.633977	?????????
Shell	0.00112501	0.018021	?????????
Merge	0.001365	0.0166621	?????????
Quick	0.00110802	0.015233	?????????

Summary

- **Selection**
- **Bubble**
- **Insertion**
- **Merge**
- **Quick**