3: Sorting I

CSE326 Spring 2002

April 1, 2002

Sorting

- Binary search is the best searching technique. . .

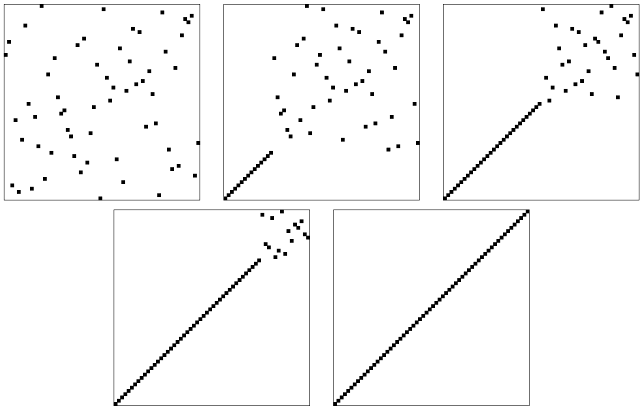
  . . . but it requires the array to be *sorted*.

- What are your favorite sorting algorithms?
  -
  -
  -
  -
  -

Selection Sort

```
void SelectSort(int *array, int n)
{
  for (int i = 0; i < n-1; i++) {
    int min = i;
    for (int j = i+1; j < n; j++)
      if (array[j] < array[min])
        min = j;
    swap array[min], array[i];
  }
}
```

Selection Sort: What It Looks Like



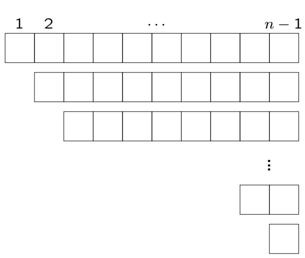
Selection Sort

- Worst-case running time?

- What is the worst-case input?

- What input does it perform well on?

Series

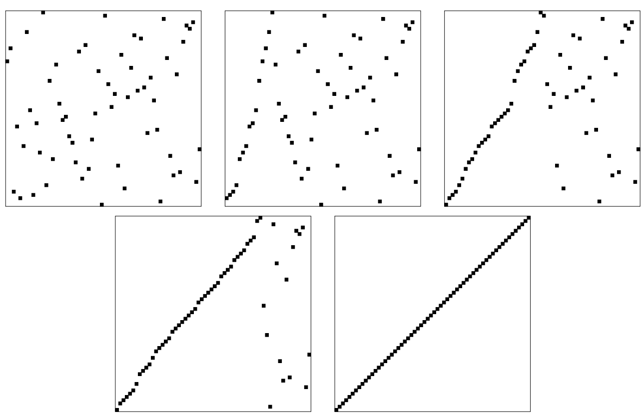


- $\sum_{1}^{n-1} i =$

Insertion Sort

```
void InsertionSort (int *array, int n)
{
  for (int i = 1; i < n; i++) {
    int x = array[i];
    for (int j = i; j > 0 && array[j-1] > x; j--)
      array[j] = array[j-1];
    array[j] = x;
  }
}
```


Insertion Sort: What It Looks Like



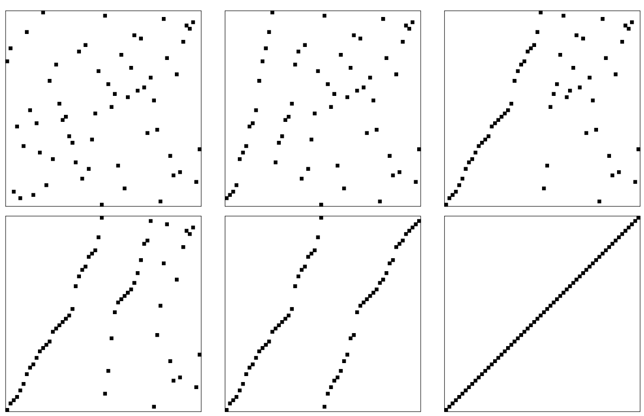
Insertion Sort

- Worst-case running time?
- What is the worst-case input?
- What input does it perform well on?

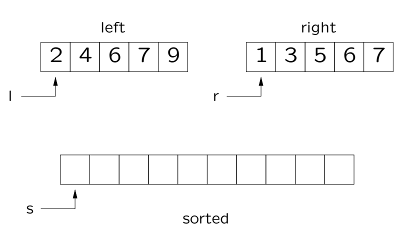
Merge Sort

```
void MergeSort(int *array, int n)
{
   if (n <= 1)
     return;

   int *buf = new int[n];
   memcpy(buf, array, sizeof(int)*n);

   int mid = n/2;

   MergeSort(buf, mid);
   MergeSort(buf+mid, n-mid);

   Merge(array, buf, mid, buf+mid, n-mid);

   delete [] buf;
}
```


Merge Sort: What It Looks Like



Merging



Merging

```
void Merge(unsigned *sorted,
           unsigned *left, unsigned llen,
           unsigned *right, unsigned rlen)
{
  unsigned l = 0, r = 0, s = 0;
  unsigned slen = llen + rlen;

  while (s < slen) {
    if (l < llen
        && (r >= rlen
            || left[l] <= right[r]))
      sorted[s++] = left[l++];
    else if (r < rlen
             && (l >= llen
                 || right[r] <= left[l]))
      sorted[s++] = right[r++];
  }
}
```


Merge Sort

- Worst-case running time?
  - Merge()?
  - MergeSort()?

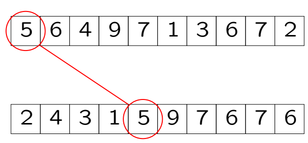
Merge Sort

- Advantages of Merge Sort?
- Disadvantages of Merge Sort?

Quicksort

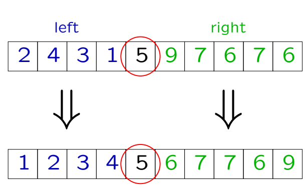
- Merge sort splits into two halves, then merges, using extra memory.
- Quicksort splits *first*, *in-place*, then combines the two halves.

Quicksort



Partitioning an array around 5

Quicksort

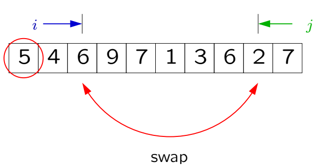


If we recursively sort the two sides of the partition, we'll sort the array!

Quicksort

```
void Quicksort(int *array, int high, int low=0)
{
  if (high > low + 1) {
    int mid = Partition(array, high, low);
    Quicksort(array, mid, low);
    Quicksort(array, high, mid+1);
  }
}
```


Partition



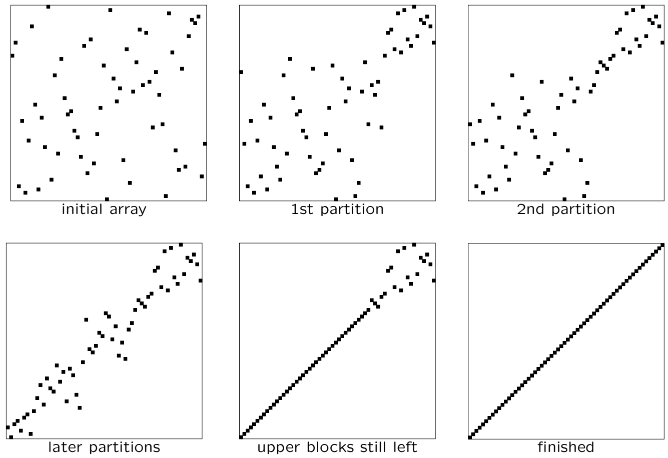
Partition about p:

- scan i from left to find array$[i] > p$
- scan j from right to find array$[j] < p$
- swap array$[i]$, array$[j]$

Partition

```
int Partition(int *array, int high, int low)
{
  int i = low;
  int j = high;
  int v = array[low];
  while (i < j) {
    i++;
    while (i < high
           && array[i] < v)
      i++;
    j--;
    while (j >= low
           && array[j] > v)
      j--;
    if (i < high)
      swap array[i], array[j];
  }
  if (i < high)
      swap array[i], array[j];
  swap array[low], array[j];
  return j;
}
```

Quicksort: What It Looks Like



Quicksort

- Worst-case running time?
  - Partition()?
  - Quicksort()?

Quicksort

- Expected-case running time?
  - Partition()?
  - Quicksort()?

Quicksort

How much space?

Improvements to Partition

- Choose partitioning element *randomly*

Improvements to Partition

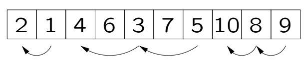
- Use *median-of-three* partitioning

Improvements to Quicksort

- What does the array look like if we stop recursing on ranges smaller than 4?

More Insertion Sort

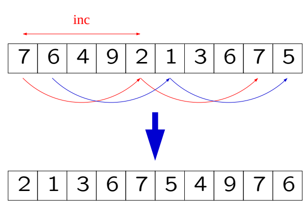
```
void InsertionSort (int *array ,
                    int n)
{
  for (int i = 1; i < n; i++) {
    int x = array [i];
    for (int j = i;
          j > 0
            && array[j-1] > x;
          j--)
      array [j] = array [j-1];
    array [j] = x;
  }
}
```



- If every element is within two hops of its final location, insertion sort is $O(n)$
- On a "nearly" sorted array, insertion sort is *linear* time

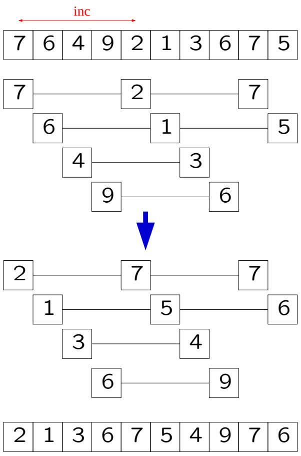
Increment Insertion Sort

```
void IncInsSort (int *array ,
                 int n,
                 int inc)
{
  for (int i = inc; i < n; i++) {
    int x = array [i];
    for (int j = i;
          j > inc
            && array[j-inc] > x;
          j -= inc)
      array [j] = array [j-inc];
    array [j] = x;
  }
}
```



- Big increment makes array "more sorted"

Increment Insertion Sort



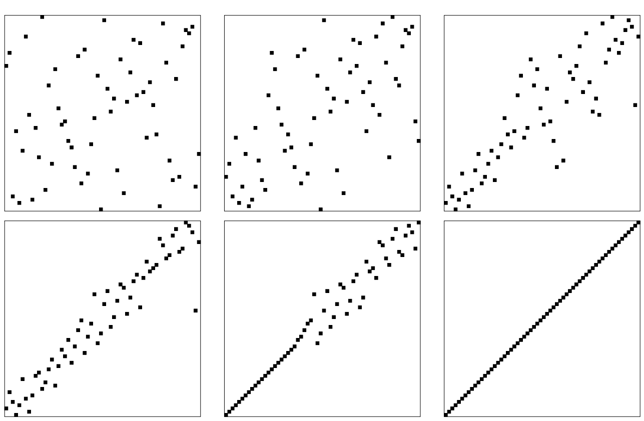
- Sorts short *interleaved* arrays simultaneously

Shell Sort

```
void IncInsSort (int *array,
                 int n,
                 int inc)
{
  for (int i = inc; i < n; i++) {
    int x = array[i];
    for (int j = i;
         j > inc
           && array[j-inc] > x;
         j -= inc)
      array[j] = array[j-inc];
    array[j] = x;
  }
}

void ShellSort (int *array, int n)
{
  for (int inc = FirstInc ();
       inc >= 1;
       inc = NextInc(inc))
    InsIncSort (array, n, inc);
}
```

- Use inc to set table size
- Start inc off large (to get table rougly sorted)
- Decrease inc (to finish up the sorting)

Shell Sort: What It Looks Like



Shell Sort: Analysis

- Unknown!
- Depends on increment sequence
  - $1, 4, 13, \ldots, 3h_{i-1} + 1, \ldots$ is worst-case $O(n^{3/2})$
    - Empirically much better
    - Conjectured to be $\Theta(n \log^2 n)$ or $\Theta(n^{5/4})$
  - Other sequences shown to be $O(n \log^2 n)$
  - Provably best sequence unknown. $\Theta(n \log n)$?