CS 4/56101

Design and Analysis of Algorithms

Kent State University

Dept. of Math & Computer Science

LECT-6

Sorting

3

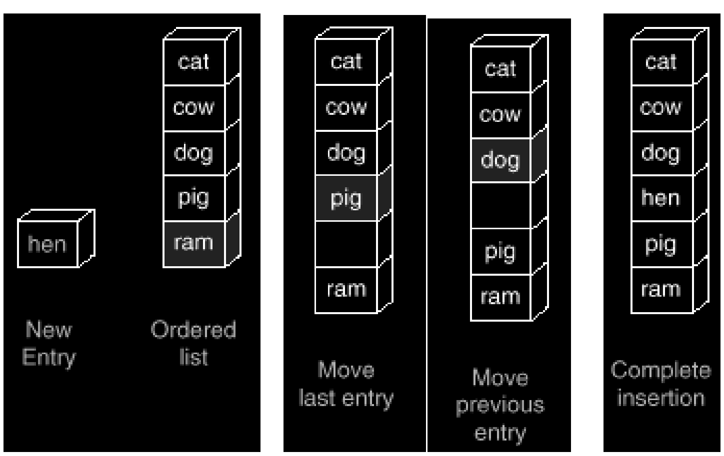
Sorting

- A little old estimate said that more than half the time on many commercial computers was spent in sorting.
- Knuth's book lists about 25 sorting methods and claims they are only fraction of the algorithms that have been devised so far.

- Types of sorting:
  - External vs. Internal

Insertion Sort

- Insertion in an Ordered List



Sorting by Insertion

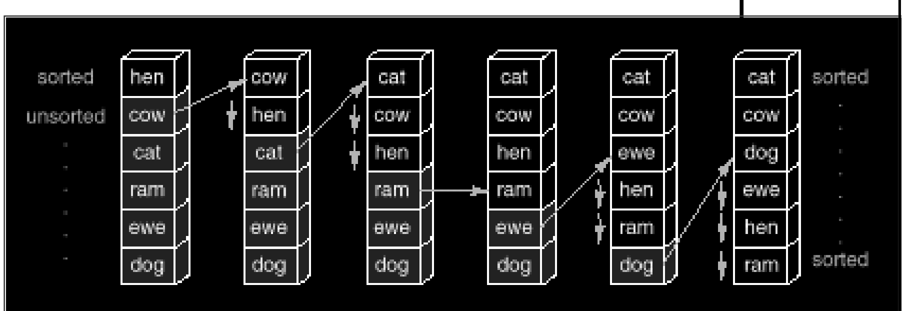
DESIGN &
ALALYSIS OF
ALGORITHM

- Maintain two lists, one sorted, another unsorted.
- Initially the sorted list has size zero, unsorted list has all the original keys.
- One by one insert the keys from unsorted list to the right position in the sorted list.

Select 6 Names and play contiguous and linked list versions! (Volunteer needed!)

Sorting by Insertion (Example)

DESIGN &
ALALYSIS OF
ALGORITHM



Insertion Sort (contiguous list)

```
void InsertionSort(List *list)
{
    Position fu;       /*first unsorted entry position*/
    Position place;    /*searches sorted part of list*/
    ListEntry current; /*holds entry temporarily*/

    for (fu = 1; fu < list->count; fu++)
        if(LT(list->entry[fu].key,list->entry[fu-1].key)){
            current = list->entry[fu];
            for (place = fu - 1; place >= 0; place--) {
                list->entry[place+1]=list->entry[place];
                if (place==0||
                LE(list->entry[place-1].key, current.key))
                        break;
            }
            list->entry[place] = current;
        }
}
```


Insertion Sort (linked list)

```
void InsertionSort(List *list)
{
    ListNode *fu;           /* the first unsorted node to be inserted */
    ListNode *ls;           /* the last sorted node (tail of sorted sublist) */
    ListNode *current, *trailing;
    if (list->head) {
        ls = list->head;    /* An empty list is already sorted. */
        while (ls->next) {
            fu = ls->next;  /* Remember first unsorted node.    */
            if (LT(fu->entry.key, list->head->entry.key)) {
                ls->next = fu->next; fu->next = list->head; list->head = fu;
                     /*Insert first unsorted at the head of sorted list.*/
            } else {                /* Search the sorted sublist. */
                trailing = list->head;
                for (current = trailing->next; GT(fu->entry.key, current->entry.key);
                    current = current->next)
                        trailing = current;
                /* First unsorted node now belongs between trailing and current. */
                if (fu == current)
                    ls = fu;
                else {
                    ls->next = fu->next; fu->next = current; trailing->next = fu;
                }
            }
        }
    }
}
```

Analysis

DESIGN &
ALALYSIS OF
ALGORITHM

- i th entry requires anywhere between 0 to (i-1) iterations. On the average it requires
  - [0+1….+(i-1)]/(i-1)= i/2 iterations
- Each iteration has
  - 1 comparison and
  - 1 assignment
- Outside the loop there are
  - 1 comparison and
  - 2 assignments
  - cost is $Comp = \frac{i}{2} + 1$

$$Assignments = \frac{i}{2} + 2$$

```
void InsertionSort(List *list)
{
    Position fu;       /*first unsorted entry position*/
    Position place;    /*searches sorted part of list*/
    ListEntry current; /*holds entry temporarily*/

    for (fu = 1; fu < list->count; fu++)
        if(LT(list->entry[fu].key,list->entry[fu-1].key)){
            current = list->entry[fu];
            for (place = fu - 1; place >= 0; place--) {
                list->entry[place+1]=list->entry[place];
                if (place==0||
                        LE(list->entry[place-1].key,
      current.key))
                        break;
            }
            list->entry[place] = current;
        }
}
```


Analysis

DESIGN &
ALALYSIS OF
ALGORITHM

- i th entry requires anywhere between 0 to (i-1) iterations. On the average it requires
  - [0+1….+(i-1)]/(i-1)= i/2 iterations
- Each iteration has
  - 1 comparison and
  - 1 assignment
- Outside the loop there are
  - 1 comparison and
  - 2 assignments
  - cost is $Comp = \frac{i}{2} + 1$

$$Assignments = \frac{i}{2} + 2$$

- i iterates from 2 to n:

- But before we proceed lets simplify using Big-O rules:

$$Comparisions = \frac{i}{2} + O(1)$$

$$Assignments = \frac{i}{2} + O(1)$$

- Total Cost:

$$= \sum_{i=2}^{n} [\frac{i}{2} + O(1)] = \frac{1}{2} \sum_{i=2}^{n} i + O(n) = \frac{1}{4} n^2 + O(n)$$

DESIGN &
ALALYSIS OF
ALGORITHM

Quiz:

When the worst case performance occurs?

When the best case performance occurs?

DESIGN &
ALALYSIS OF
ALGORITHM

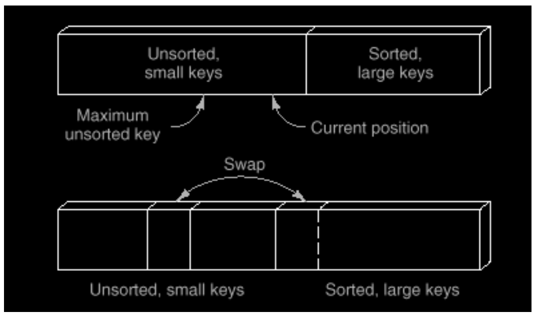
Comments on Insertion Sort

- Insertion sort is an excellent method to check if a sorted list is still sorted.
- It is also good if a list is nearly in order.
- The main disadvantage of insertion sort is that there are too many moves, even on sorted keys, if just one key is out of place.
- A data which needs to travel at far away location needs to go through many steps.
- One data moves just one position in one iteration.

Selection Sort

DESIGN & ALALYSIS OF ALGORITHM

- Selection sort one by one selects the max (or min) keys from the unsorted list and just appends them at the end of the sorted list.
- Consequently, there is no insertion cost.



Selection Sort (Contiguous list)

DESIGN & ALALYSIS OF ALGORITHM

```
void SelectionSort(List *list)
{
    Position current; /*position of place being
    correctly filled*/
    Position max;     /*position of largest remaining
    key */

    for (current = list->count - 1; current > 0;
    current--) {
        max = MaxKey(0, current, list);
        Swap(max, current, list);
    }
}
```

Selection Sort (Contiguous list)

```
Position MaxKey(Position low, Position high, List *list)
{
    Position largest;       /* position of largest key so far */
    Position current;       /* index for the contigous list */

    largest = low;
    for (current = low + 1; current <= high; current++)
        if (LT(list->entry[largest].key, list->entry[current].key))
            largest = current;
    return largest;
}
```

```
void Swap(Position low, Position high, List *list)
{
    ListEntry temp = list->entry[low];

    list->entry[low] = list->entry[high];
    list->entry[high] = temp;
}
```

DESIGN & ALALYSIS OF ALGORITHM

Analysis

- Swap is called n-1 times
  - each has 3 assignments
- MaxKey is called n-1 times. Length t of the sub list varies from n to 2.
  - Each requires t-1 comparisons.
  - Total 3(n-1) assignments.
- Thus there are:
  - Thus (n-1)+(n-2)+….+1
  - =.5 n (n-1) comparisons.

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

DESIGN & ALALYSIS OF ALGORITHM

Comparison of Selection and Insertion Sort

DESIGN & ALALYSIS OF ALGORITHM

	Selection	*Insertion (average)*
Assignments of entries	$3.0n + O(1)$	$0.25n^2 + O(n)$
Comparisons of keys	$0.5n^2 + O(n)$	$0.25n^2 + O(n)$

- Quiz:
- What is the best case for selection sort?
- What is the worst case for selection sort?
- Which method should we use
  - For large n?
  - If we know, the list is almost sorted?
  - Cost of assignment is large?

Selection sort is good at moving but bad at comparison. Insertion sort is just opposite!

Can we combine both the good qualities?

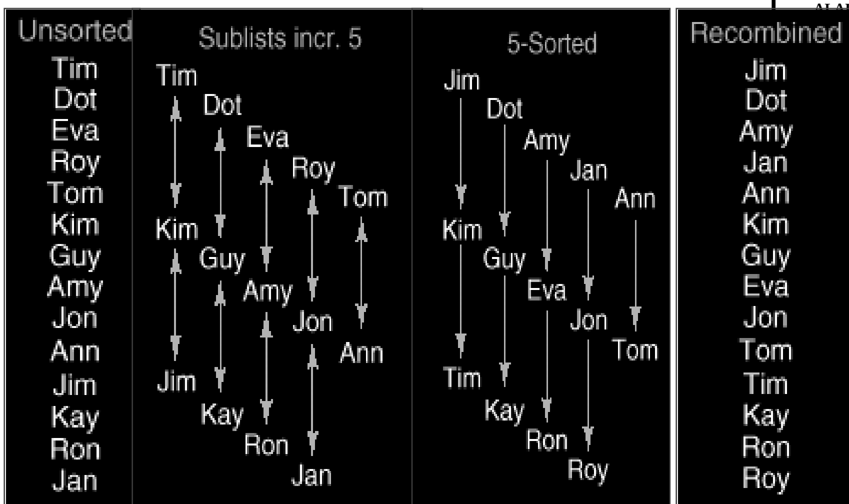
Shell Sort

DESIGN & ALALYSIS OF ALGORITHM

- The problem with insertion sort is that, if a data needs to move much long distance it have to go through many iterations.

- Solution is Shell Sort!
- Invested by D.L. Shell in 1959.

Idea (Step-1)

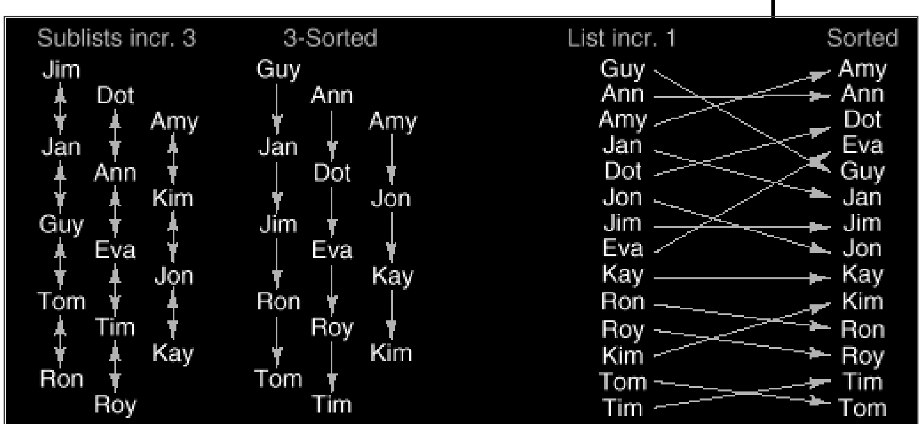


DESIGN & ALALYSIS OF ALGORITHM

LECT-06, S-20
ALG00S, javed@kent.edu
Javed I. Khan@1999

Step-2

DESIGN & ALALYSIS OF ALGORITHM



LECT-06, S-21
ALG00S, javed@kent.edu
Javed I. Khan@1999

Shell Sort

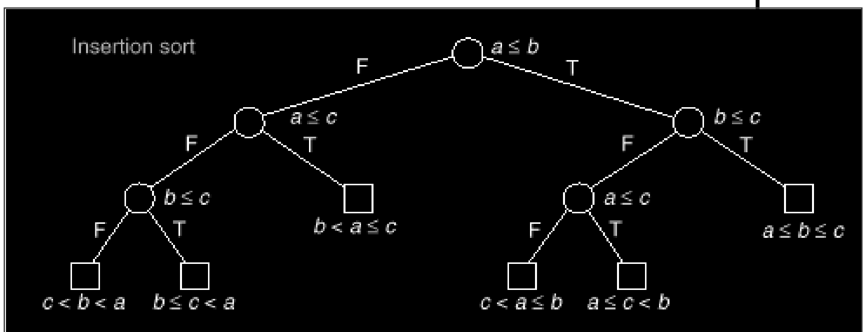
- How to select the increments?
  - 5,3,,1 worked. Many other choices will work also.
- However, no study so far could conclusively prove one choice is better that the other.
- Only requirement is that last round should be of increment 1 (that's an pure insertion sort).
- Probably it in not a good idea to use increments in power's of 2. Why?
- Analysis:
  - exceedingly difficult
  - for large n it appears the number of moves is in $^{n1.25}$ to 1.6 n $^{1.25}$.

DESIGN & ALALYSIS OF ALGORITHM

Lower Bounds of Sorting

Comparison Tree of Insertion Sort (a,b,c)

DESIGN & ALALYSIS OF ALGORITHM

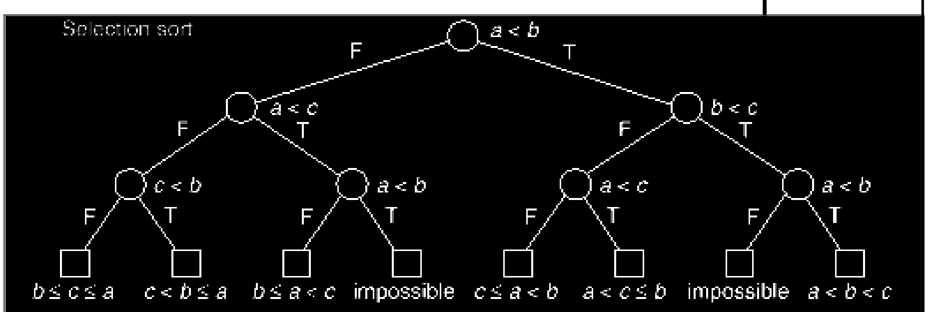


- The worst path is the worst case performance.
- The average path is the average performance.

LECT-06, S-24
ALG00S, javed@kent.edu
Javed I. Khan@1999

Comparison Tree of Selection Sort (a,b,c)

DESIGN & ALALYSIS OF ALGORITHM



- Selection sort tree is more bushy on the average.

LECT-06, S-25
ALG00S, javed@kent.edu
Javed I. Khan@1999

Limits of Sorting Algorithms

DESIGN & ALALYSIS OF ALGORITHM

- If there are n numbers to sort how many possible outcomes?

THEOREM 7.2 Any algorithm that sorts a list of n entries by use of key comparisons must, in its worst case, perform at least $\lceil \lg n! \rceil$ comparisons of keys, and, in the average case, it must perform at least $\lg n!$ comparisons of keys.

- Sterling's approximation of n!:

log e=1.442

$$\log n! \approx (n+\frac{1}{2})\log n-(\log e)\cdot n+\log\sqrt{2\pi}+\frac{\log e}{12n}$$

$$\log n! \approx (n+\frac{1}{2})(\log n-1.5)+2$$

$$= n.\log n-1.44n+O(\log n)$$

LECT-06, S-26
ALG00S, javed@kent.edu
Javed I. Khan@1999

Next Class:
Quick & Merge Sort

27