A Course on Algorithm

Design & Analysis

CS 4/56101

Design and Analysis of Algorithms

Kent State University

Dept. of Computer Science

LECT-8

Random Variable & Binomial Distribution

Random Variable

- A random variable is a function from a finite or countably infinite sample space S to the real numbers. It associates a real number with each possible outcome of an experiment.
- Generally we associate a probability value with each possible outcome of X. The function describing the distribution is called probability distribution function.
- Example (Binomial Distribution):
  - We are tossing a coin.
  - We score x_i=1 for head and 0 for tail.
  - Each time the probability of head is p.
  - If we toss n times, what is the probability the score X= $x_1+x_2+x_3+x_n$ will be exactly 2, 3, ..k or n?

$$\Pr\{X = k\} \\ = f(n, k, p)$$

Binomial Distribution

- We are tossing a coin n times. Each time the probability of head is p. What is the probability that X, exactly has the value k? $\binom{n}{k}$
  – the ways of picking k heads
  – the probability of occurring each $p^k(1-p)^{n-k}$
  – Binomial probability distribution is written as:

$$\Pr\{X=k\} = b(k;n,p) = \binom{n}{k} p^k (1-p)^{n-k}$$

  – The name "binomial" comes from the fact the terms of the expansion of

$$(p+q)^n, \ when \quad p+q=1$$

  – A relation:

$$\sum_{k=0}^{n} b(k;n,p) = \sum_{k=1}^{n} \binom{n}{k} p^k (1-p)^{n-k} = (p+q)^n = 1$$

DESIGN & ALALYSIS OF ALGORITHM

Expectation & Variance of X

- Expectation:

$$E[X] = \sum_{k=0}^{n} kb(k:n,p) = np$$

- Variance:

$$Var[X] = E[(X-E[X])^2] = npq$$

DESIGN & ALALYSIS OF ALGORITHM

Expectation of X (derivation)

- Expectation:

$$E[X] = \sum_{k=0}^{n} kb(k:n,p)$$
$$= \sum_{k=1}^{n} k\binom{n}{k} p^k.q^{n=k} = \sum_{k=1}^{n} k.\frac{n!}{k!(n-k)!} p^k.q^{n-k}$$
$$= np.\sum_{k=1}^{n} \frac{n-1!}{(k-1)!(n-k)!} p^{k-1}.q^{n-k}$$
$$= np.\sum_{k'=0}^{n'} \frac{n'!}{k'!(n'-k')!} p^{k'}.q^{n'-k'}$$
$$= np\sum_{k'=0}^{n'} b(k':n',p) = np$$

Variance of X (derivation)

$$E[X^2] = E[(X_1 + X_2 + X_3 + .. + X_n)^2]$$
$$= \sum_{i=1}^{n} E[X^2{}_i] + 2\binom{n}{2} E[X_i].E[X_i]$$
$$= np + 2\frac{n(n-1)}{2} p^2 = np + n^2p^2 - np^2$$

$$Var[X] = E[(X - E[X])^2]$$
$$= E[X^2] - E[X].E[X]$$
$$= np + (np)^2 - np^2 - (np)^2$$
$$= np(1-p) = npq$$

Counting Sort

DESIGN &
ALALYSIS OF
ALGORITHM

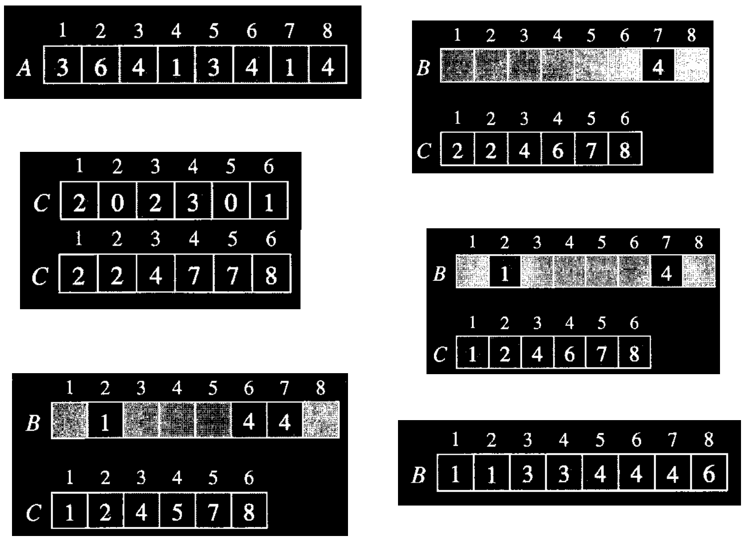
Idea

- The range of possible keys are known
- all the key are integers in the range 1 to k.

- They can be sorted in O(n) time!

- Find out the final position of each key.
- Move them there.
- Since, there might be some duplication do some extra tricks..

Example (range 1-6)



DESIGN & ALALYSIS OF ALGORITHM

Code

```
COUNTING-SORT(A, B, k)
1   for i ← 1 to k
2        do C[i] ← 0
3   for j ← 1 to length[A]
4        do C[A[j]] ← C[A[j]] + 1
5   ▷ C[i] now contains the number of elements equal to i.
6   for i ← 2 to k
7        do C[i] ← C[i] + C[i − 1]
8   ▷ C[i] now contains the number of elements less than or equal to i.
9   for j ← length[A] downto 1
10       do B[C[A[j]]] ← A[j]
11            C[A[j]] ← C[A[j]] − 1
```

L1-2:Initialization

L3-4: Counting

L6-7: Position

L10-11: Move

L10-11: Update next's position

DESIGN & ALALYSIS OF ALGORITHM

DESIGN & ALALYSIS OF ALGORITHM

Complexity

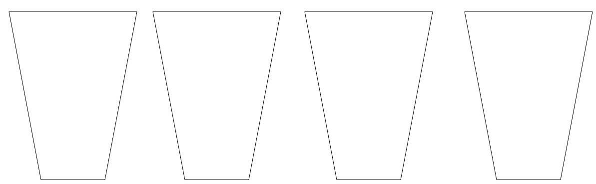
- Analysis:
  - K = possible types of keys
  - n= number of keys
  - L:1-2: Initialization O(k)
  - L:3-4: Counting O(n)
  - L:6-7: Position offset O(k)
  - L:9-11: Moves O(n)
- Overall time complexity O(n+k)
- A Definition: Stability of a Sorting Algorithm
  - Counting sort is stable.
  - Quick sort is not.

```
COUNTING-SORT(A, B, k)
1   for i ← 1 to k
2       do C[i] ← 0
3   for j ← 1 to length[A]
4       do C[A[j]] ← C[A[j]] + 1
5   ▷ C[i] now contains the number
6   for i ← 2 to k
7       do C[i] ← C[i] + C[i − 1]
8   ▷ C[i] now contains the number
9   for j ← length[A] downto 1
10      do B[C[A[j]]] ← A[j]
11         C[A[j]] ← C[A[j]] − 1
```


Radix Sort

Idea

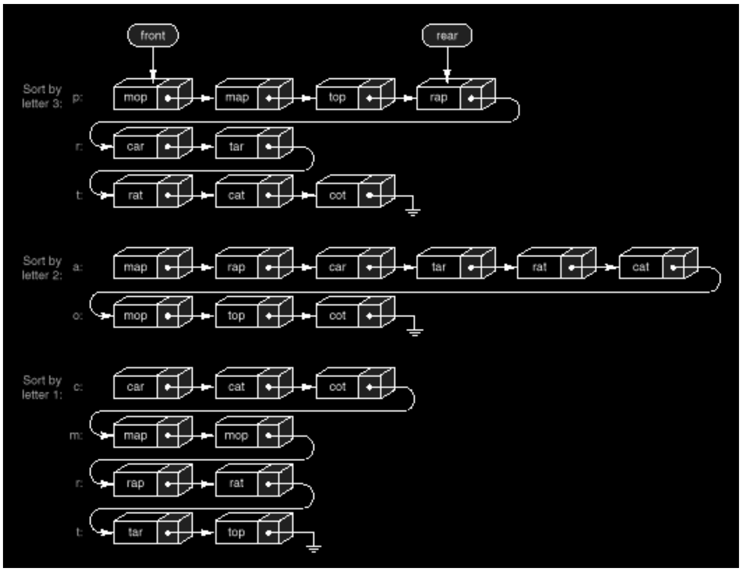
- Have a number of buckets.
- Sort one alphabet at a time.
- Stack them up.
- Sort next key!



Example

Initial order	Sorted by letter 3	Sorted by letter 2	Sorted by letter 1
rat	mop	map	car
mop	map	rap	cat
cat	top	car	cot
map	rap	tar	map
car	car	rat	mop
top	tar	cat	rap
cot	rat	mop	rat
tar	cat	top	tar
rap	cot	cot	top

Linked Radix Sort



The Main Function

```
void RadixSort(List *list)
{
    int     i, j;
    Node    *x;
    Queue   queues[MAXQUEUEARRAY];

    for (i = 0; i < MAXQUEUEARRAY; i++)
        CreateQueue(&queues[i]);
    for (j = KEYSIZE - 2; j >= 0; j--) {
        while(!ListEmpty(list)) {
            DeleteList(0, &x, list);
            AppendNode(x, &queues[QueuePosition(x-
>entry[j])]);
        }
        Rethread(list, queues);
    }
}
```


DESIGN &
ALALYSIS OF
ALGORITHM

```
/* QueuePosition:  determine the queue position (0 through 27)
    for a character.*/
int QueuePosition(char c)
{
    if (c == ' ')
        return 0;
    else if (isalpha(c))
        return tolower(c) - 'a' + 1;
    else
        return 27;
}
```

```
/* Rethread:  rethread a list from an array of
queues.*/
void Rethread(List *list, Queue queues[])
{
    int     i;
    Node    *x;
    for (i = 0; i < MAXQUEUEARRAY; i++)
        while(!QueueEmpty(&queues[i])) {
            ServeNode(&x, &queues[i]);
            InsertList(ListSize(list), x, list);
        }
}
```


Complexity

- N is the number of keys
- k is the max length of the key.
- Time complexity is O(nk).
- The best- Merge Sort was n log n!

- If k is large, but only few key are long Merge sort will be better.

- Is Radix Sort Stable?
  - So long as the single alphabet sort is.

Any difference between a comparison in quick sort and a comparison in radix sort?