Sorting

Sorting

- Sorting refers to arranging a set of data in some logical order.
- Example: Telephone directory having name, address, and phone number.
- Example: 1, 2, 3, 4, 5, 6, 7, 8, 9

Hou F L Dr phys & surg 1530 W 7th ---- 738-2151
Hou James C M 1292 W 49th ---------- 261-1537
Hou John 104-1972 Robson ---------- 688-7576
Hou John C Dr dent 6035 Sussex Bby ---- 437-6444
Hou T L 102-1140 Pendrell ---------- 687-1287
Houang K T 759 E 22nd ------------ 872-6921
Houang Yong Ning 2431 E Georgia ----- 254-0269
Houben A M 951 Ringwood ---------- 879-9953
Houben Robert 202-436-7th St N West -- 524-1482
Houchen Gordon K 1A-1366 W 14th ---- 738-7021
Houchen Jerry 10206-127th St Sry ----- 581-4834
Houchen K R 15104-92A Ave Sry ------ 588-2696
Houchen Leo 10237-123A St Sry ------ 584-3645
Houchen M 606-1945 Barclay -------- 688-3065
Houchen Wm
318-13270-105A Ave Sry --------- 584-1419
Houchins S V 261 E 17th ----------- 874-0575
Houck C 15761 Cliff W Rock --------- 536-5528
Houck Donald A 575 E St James NVn ---- 988-7578
Houck L F 2036 Berkley NVn --------- 929-1785

Why to Sort?

- Searching an element is faster.
- Binary search $\log_2 n$ vs Linear search n.
- Rank Queries: find the kth largest/smallest value.

1 2 3 4 5 6 7 8 9

If array is sorted!

- It is difficult to insert an element while preserving the sorted structure.

40 50 55 60 70 75 80 85 90 92
Insert 65

- Similarly, it is difficult to delete an element while preserving the sorted structure.

40 50 55 60 70 75 80 85 90 92
Delete 70

Sorting Algorithms

- Selection Sort
- Bubble Sort
- Merge Sort
- Quick Sort

What is the difference? Different techniques, different runtimes,....

Selection Sort

- Consider the array: 6 4 9 1 6 2 0

You need to sort it in decreasing order: 9 6 6 4 2 1 0

- Approach:
  - What is the location of largest element in the sorted array?
  - What is the location of second largest element in the sorted array?

- As the name suggests, you need to select something.
- Select the largest element in your array and swap it with the first element of the array.
- Next, consider the subproblem of finding largest element in the array having n-1 elements, and so on.
- Finally, a single element is already sorted.

Code

```
void selection_sort(int a[], int start, int end) {
        if(start == end)
                return;
        int pos = findMax(a, start, end);
        swap(a, pos, start);
        selection_sort(a, start+1, end);
}
void swap(int a[], int i, int j) {
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
}
```

Bubble Sort

- Consider the array: 5 3 1 9 8 2 4 7

You need to sort it in increasing order: 1 2 3 4 5 7 8 9

- Approach:
  - Exchange consecutive values that are not in the correct order.
  - Observe the first iteration of the strategy.

Example

i = 0	j	0	1	2	3	4	5	6	7
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i = 1	0	3	1	5	8	2	4	7	9
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i = 2	0	1	3	5	2	4	7	8	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i = 3	0	1	3	2	4	5	7		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i = 4	0	1	2	3	4	5			
	1	1	2	3	4				
	2	1	2	3	4				
i = 5	0	1	2	3	4				
	1	1	2	3					
i = 6	0	1	2	3					
		1	2						

Code

```
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// An optimized version of Bubble Sort
void bubbleSort(int arr[], int n)
{
   int i, j, swapped;
   for (i = 0; i < n-1; i++)
   {
     swapped = 0;
     for (j = 0; j < n-i-1; j++)
     {
        if (arr[j] < arr[j+1])
        {
           swap(&arr[j], &arr[j+1]);
           swapped = 1;
        }
     }

     // IF no two elements were swapped by inner loop,
then break
     if (swapped == 0)
        break;
   }
}
```

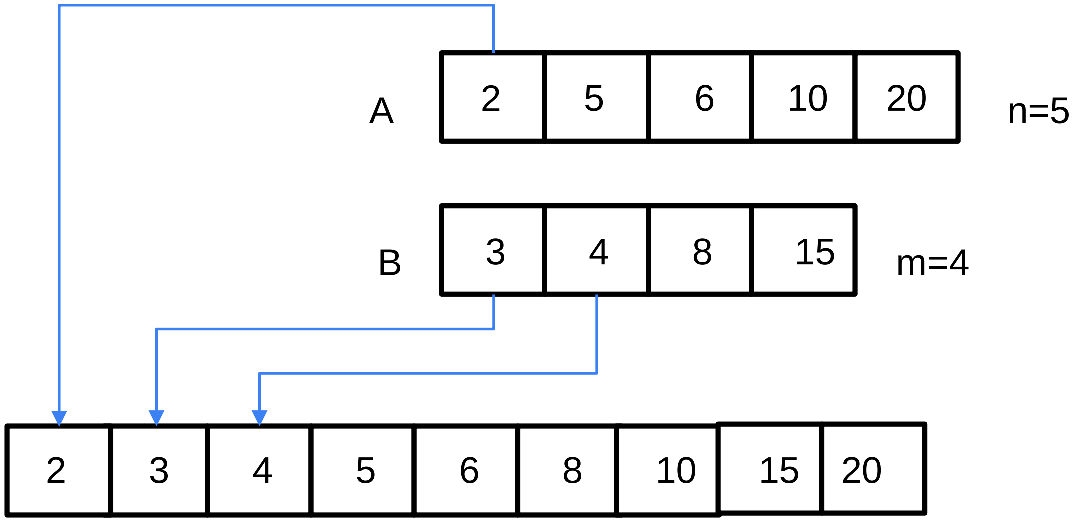
Merge Sort

- Merge sort consists of 3 parts.
    - Divide an array into two parts, A and B.
    - Sort array A and B independently (recursive call).
    - Merge A and B.

```
void merge_sort(int ar[], int start, int n) {
        if (n>1) {
                int half = n/2;
                merge_sort(ar, start, half);
                merge_sort(ar, start+half, n-half);
                merge(ar, start, n);
        }
}
```

Merging Two Sorted Arrays

- Merge two sorted arrays A of size n and B of size m.
- Create an empty array C of size n + m.
- Variable i, j, and k



Merge Code

```
void merge(int ar[], int start, int n) {
        int temp[50], k, i = start, j = start + n/2;
        int lim_i = start + n/2, lim_j = start + n;
        for(k = 0; k<n; k++) {
                if((i < lim_i) && (j < lim_j)) {
                        if(ar[i] <= ar[j]) { temp[k] = ar[i]; i++;}
                        else { temp[k] = ar[j]; j++; }
                }
                else if(i == lim_i) {
                        temp[k] = ar[j]; j++;
                }
                else {
                         temp[k] = ar[i]; i++;
                }
        }
        for (k=0; k<n; k++)
                ar[start+k] = temp[k];    //in-place sorting
}
```

Quick Sort

- Partition the array into two parts using a pivot, call it “a”.
    - One part contains all elements smaller than a
    - Other part contains all elements larger than a
    - Sort the partitions recursively

- A pivot is any integer chosen to partition the array
    - It can be chosen randomly
    - It can be decided arbitrarily, say first element of the array

Partition function

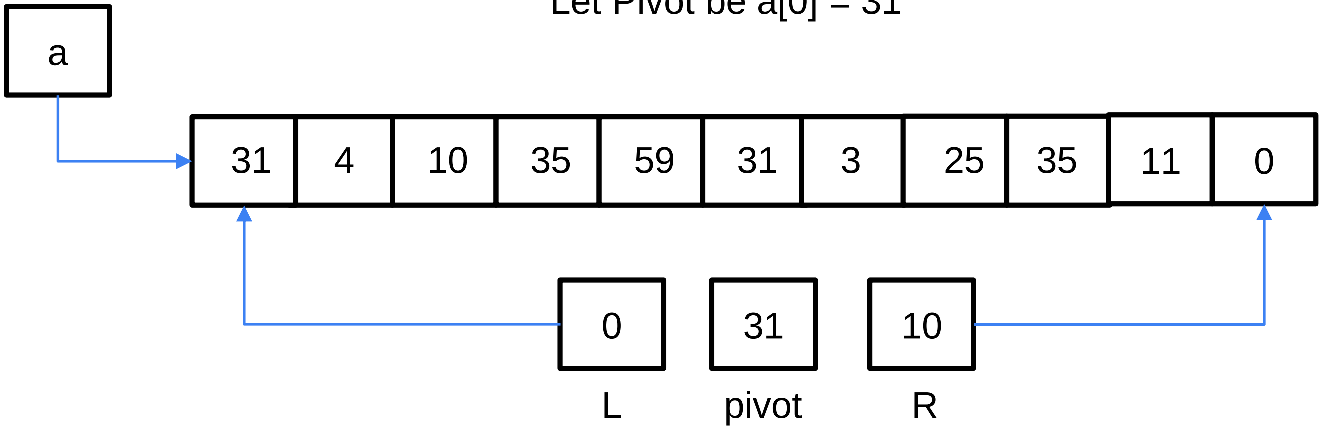
- Partition takes an array a[] of size n and a value called the pivot.
- Pivot: is an element in the array, usually chosen as a[0].
- Partition re-arranges the array elements into two parts:
  - the left part has all elements <= pivot
  - the right part has all elements >= pivot
- Partition returns the index of the beginning of the right part.

31	4	10	35	59	31	3	25	35	11

Let Pivot be a[9] = 11

Steps

Let Pivot be a[0] = 31



As long as a[L] <= pivot, advance L by 1.
As long as a[R] >= pivot, decrease R by 1.
If L < R, exchange a[L] with a[R]. Advance L by 1; decrease R by 1.

Code

```
int partition (int arr[], int low, int high)
{
    int pivot = arr[high];  // selecting last element as pivot
    int i = (low - 1);  // index of smaller element

    for (int j = low; j <= high- 1; j++)
    {
        // If the current element is smaller than or equal to pivot
        if (arr[j] <= pivot)
        {
            i++;    // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

Quick Sort

```
void quicksort(int a[], int p, int r)
{
    if(p < r)
    {
        int q;
        q = partition(a, p, r);
        quicksort(a, p, q-1);
        quicksort(a, q+1, r);
    }
}
```