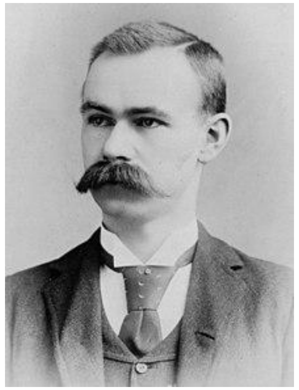
Week 13:

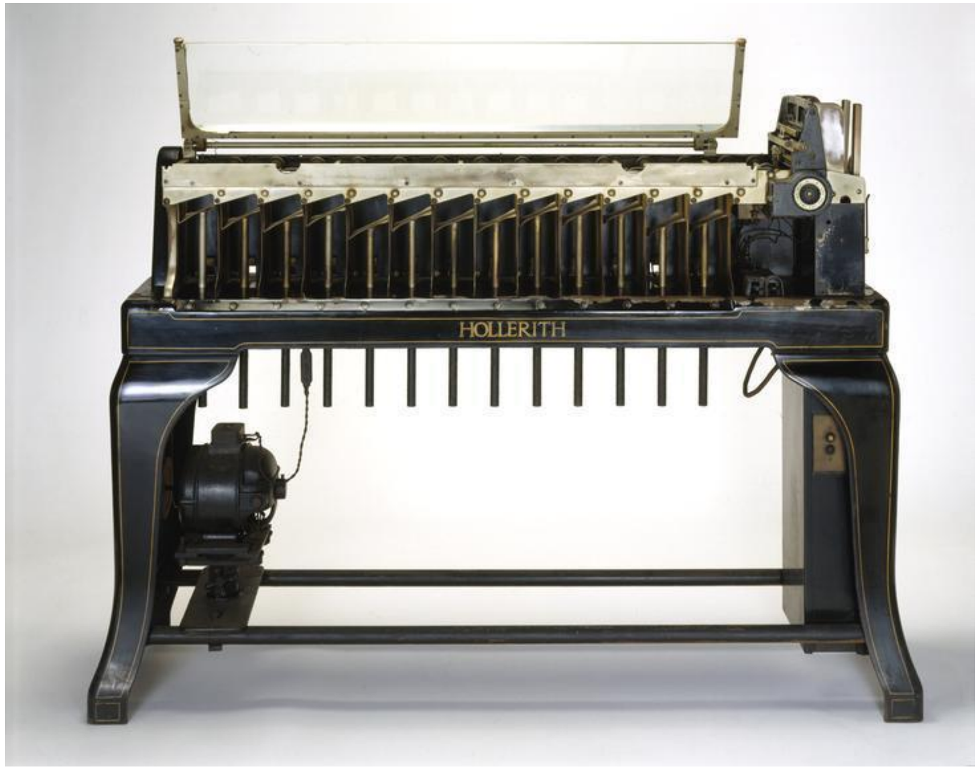
- Course admin
- Sorting
  - Behold a O(nlogn) algorithm!
- Searching

Sorting: A bit of history

- Herman Hollerith's sorting machine developed in 1901-1904 used radix sort. Punched card sorter
  - First the 0s pop out, then 1s, etc.
  - For 2-column numerical data would sort of units column first, then re-insert into machine and sort by tens column
- Patented the Electric Tabulating Machine
- His company became IBM



IBM



Selection Sort

Bubble Sort

Merging two sorted lists

1	1	2	5	8

3	4	6

1	1	2	3	4	5	6	8

Steps

1. Split the stack into two little stacks
2. Give one stack to one person, wait for the result
3. Give the other stack to another person, wait for the result
4. Place both stacks in front of you, with the top card of each stack visible.
5. Look at the top card from each stack.
6. Pick the smaller card and place it face down onto a new pile (this will become your merged stack).
7. Repeat steps 2–4 until one of the stacks is empty.
8. Once one stack is empty, take the rest of the cards from the other stack and place them on the merged stack in order, one by one.
9. Flip the merged stack over so the smallest card ends up on top.

```python
def merge(leftL, rightL):
    sortedList = []
    i = 0
    j = 0
    while i < len(leftL) and j < len(rightL):
        if leftL[i] < rightL[j]:
            sortedList.append(leftL[i])
            i+=1
        else:
            sortedList.append(rightL[j])
            j+=1
    if i < len(leftL):
        sortedList += leftL[i:]
    if j < len(rightL):
        sortedList += rightL[j:]
    return sortedList
```

Mergesort

Searching on sorted data

- Binary search is an efficient algorithm for finding an element from a sorted collection of items.
- Example: Find an element **e** in a Python list **L**
- Easy: `e in L` efficiency?
- If **L** is sorted, we can do much better
- How?

Binary Search: Recursive

```python
def binarySearch(L, x):
    # Check base case
    if len(L) == 0:
        # Element is not present in the empty list
        return False
    else:
        mid = len(L) // 2
        # If element is present at the middle itself
        if L[mid] == x:
            return True
        # If element is smaller than mid, then it can only
        # be present in left half
        elif L[mid] > x:
            return binarySearch(L[:mid], x)

        # Else the element can only be present in right half
        else:
            return binarySearch(L[mid+1:], x)
```

Bad implementation, why?

```python
def binarySearch(L, low, high, x):
    if low > high:
        return False
    mid = (high + low) // 2
    if x == L[mid]:
        return True
    elif x < L[mid]:
        return binarySearch(L, low, mid-1, x)
    else:
        return binarySearch(L, mid+1, high, x)
```

Now it is OK, why?

Binary Search: Iterative

```python
def binarySearch(L, x):
    low = 0
    high = len(L)-1
    while low <= high:  # while the interval is not empty
        mid = (high + low) // 2
        # If element is present at the middle itself
        if L[mid] == x:
            return True
        # If element is smaller than mid, then it can only
        # be present in left half
        elif x < L[mid]:
            high = mid-1
        # Else the element can only be present in right half
        else:
            low = mid+1
    return False
```

Binary Search: other uses

Write the function `lowerBound(L, x)` that returns the largest element of L that is strictly less than x. Assume no duplicates. **The function must be O(logN)**

`Examples:`

```
assert(lowerBound([4,7,9,11,12,13,15,16,17], 4) == None)
assert(lowerBound([4,7,9,11,12,13,15,16,17], 5) == 4)
assert(lowerBound([4,7,9,11,12,13,15,16,17], 6) == 4)
assert(lowerBound([4,7,9,11,12,13,15,16,17], 7) == 4)
assert(lowerBound([4,7,9,11,12,13,15,16,17], 8) == 7)
assert(lowerBound([4,7,9,11,12,13,15,16,17], 20) == 17)
```

Challenge: countInRange(L)

- Write the function `countInRange(L, a, b)` that returns the number of elements in the open range (a, b), that is, numbers between a and b, both bounds exclusive. **The function must be O(logN)**

- `L = [4,7,9,11,12,13,15,16,17]`
- `assert(countInRange(L, 4, 7)==0)`
- `assert(countInRange(L, 4, 12)==3)`
- `assert(countInRange(L, 4, 20)==8)`
- `assert(countInRange(L, 1, 3)==0)`
- `assert(countInRange(L, 12, 14)==1)`