Searching, Sorting and Alignment

Genomics and Databases

Aishwarya Alex

LCSB

UNIVERSITÉ DU LUXEMBOURG

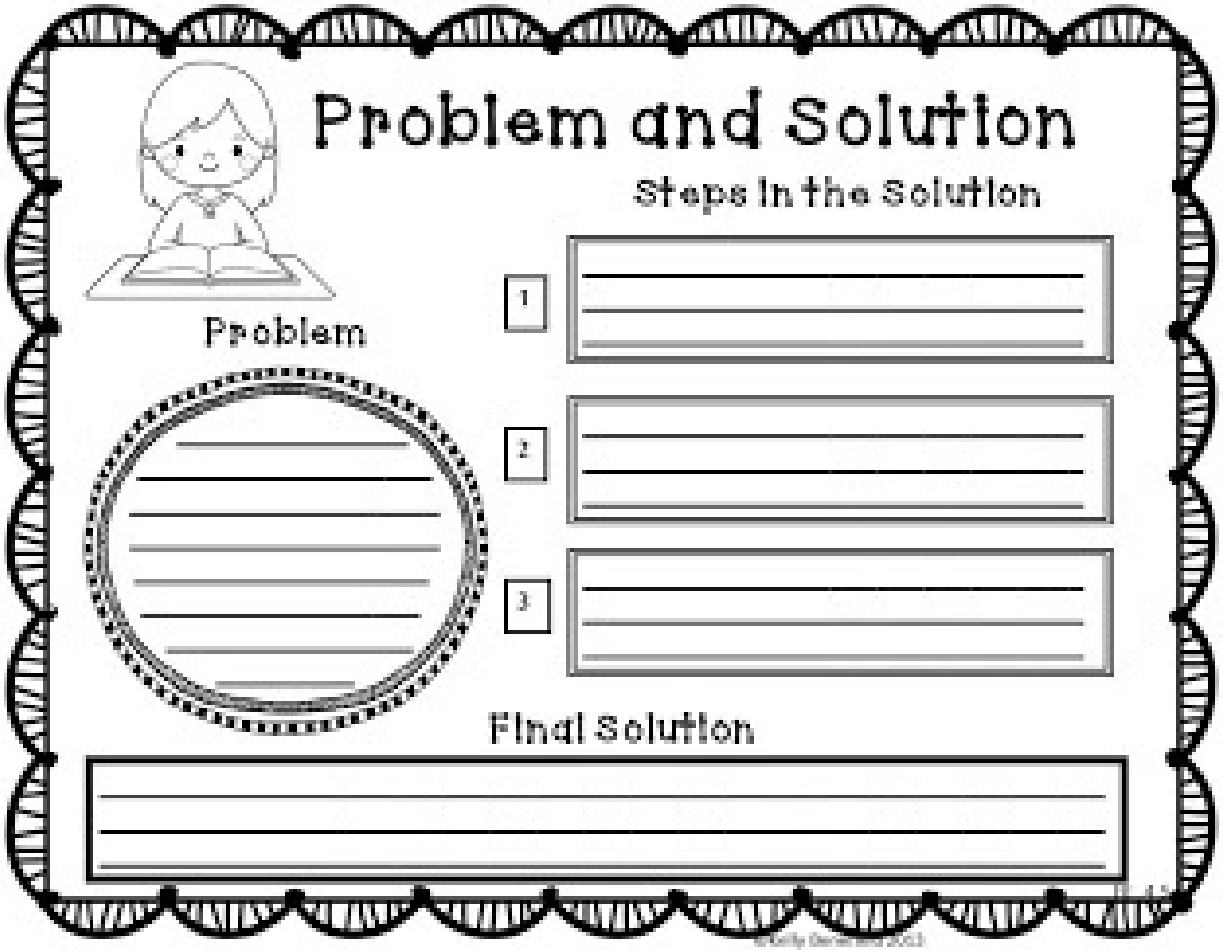
Outline

- Introduction to Algorithms and Complexity
  - What and why?
  - Complexity
- Searching
  - Exhaustive
  - Branch and bound
- Sorting
  - Bubble sort
  - Insertion sort
  - Quick sort
- Alignment
  - Manhattan Tourist Problem
  - Local Alignment
  - Global Alignment
  - BLAST

Learning objectives

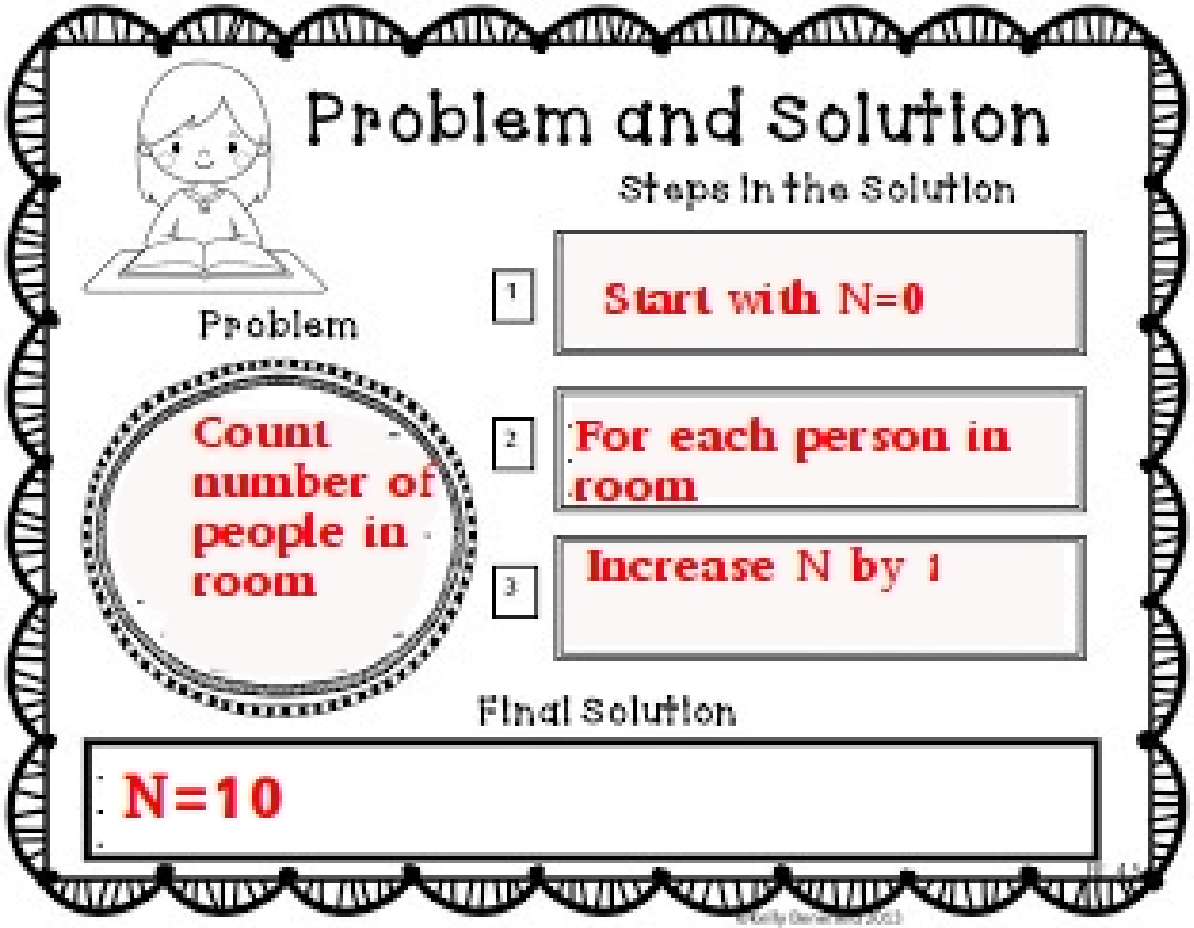
- Understand the concept of algorithm complexities
- Follow a few searching and sorting methods and the need of different methods
- Understand basic alignment techniques and their purposes

What is an algorithm?



- Steps to solve a problem

What is an algorithm?



- Several ways to the same solution
- Program= Algorithm + Data Structures

Why Algorithms?

- Same solution can be attained from different algorithms
- Independent of programming languages
- Compare algorithm space and time complexity to find the best way for current problem
- Best algorithm : least memory and fastest running time

Algorithm complexity

- Time and space required by the algorithm

Example: Counting number of people (10)

Number of iteration	Count individually	Count in pairs
1	1	2
2	2	4
3	3	6
4	4	8
5	5	10
6	6	
7	7	
8	8	
9	9	
10	10	

Search Algorithms

1. Linear search / Brute force / Exhaustive search
   - Start at the beginning and proceed examining every alternative
   - Stop only when you find the element or have no alternatives remaining
   - *List need not be ordered*
2. Binary search / Branch and bound
   - Eliminate alternatives as you proceed
   - Will not examine all alternative even if search return negative result
   - *Exploit the ordering of the list*

LCSB

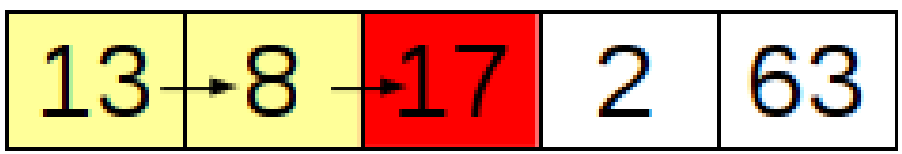
UNIVERSITÉ DU LUXEMBOURG

Linear search / Brute force / Exhaustive search

- Input list need not be sorted

Algorithm :Search **a** in target list

1. Start from first element
2. Is element = **a**; then stop (Successful)
3. Proceed to next element and repeat from 2
4. Stop if end of list (Unsuccessful)



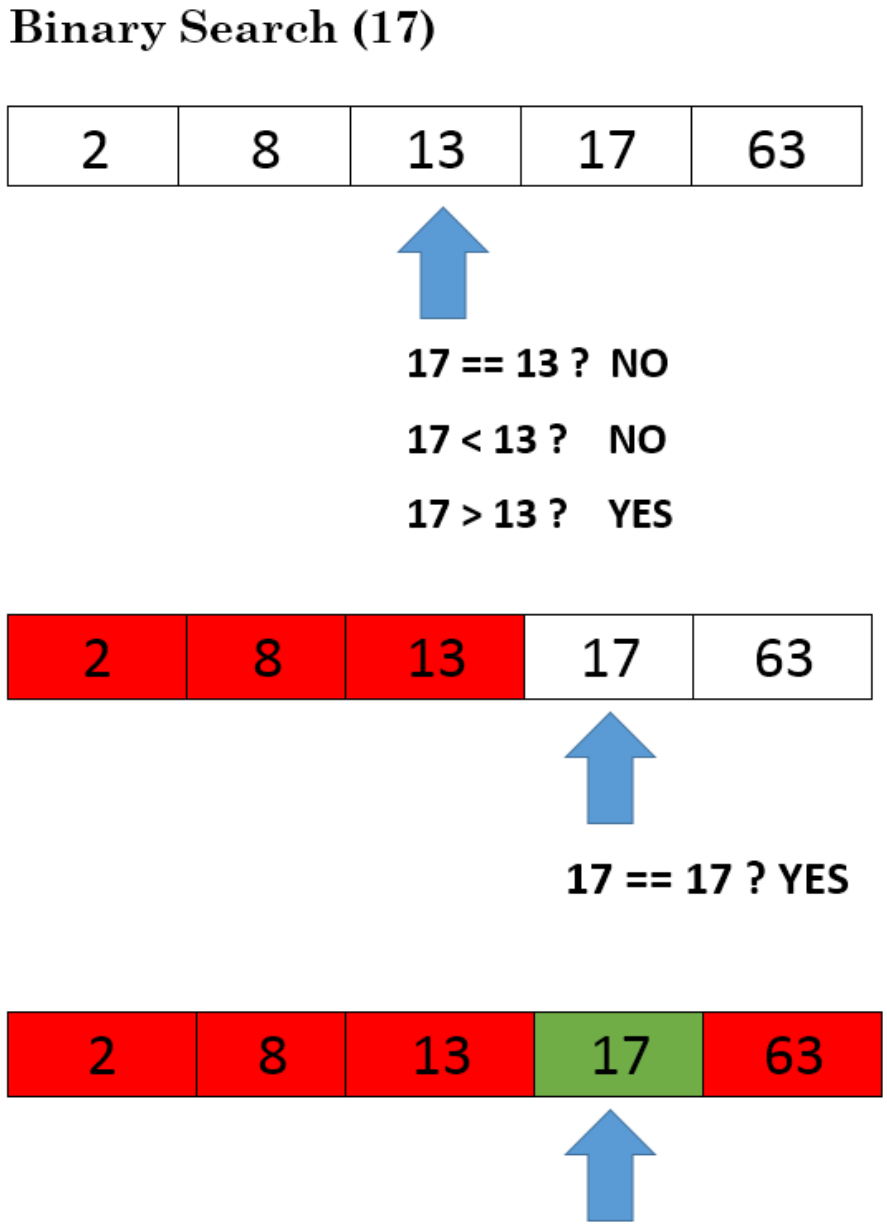
Binary search / Branch and Bound

- Input : Sorted list

Algorithm: Find **a** in target list

1. Get middle element
2. If middle element = **a** then stop (Successful)
3. Otherwise
   - 3.1 If **a** less than middle element; then repeat from step 1 with target as beginning to middle element
   - 3.2 if **a** greater than middle element; then repeat from step 1 with target as middle element to end of list
4. Stop if sublist has no more elements (Unsuccessful)

Binary search example



Sorting Algorithms

There are several sorting algorithms available. We will discuss only 3 of them.

1. Bubble sort
2. Selection sort
3. Quick sort

Bubble sort

- Compare two consecutive items in a list; swap if out of order
- Largest element *bubbles* down the list to its final position.
  i.e. After each pass of the list the largest remaining item is placed in its proper order.

Algorithm

1. Compare each element(except last) to its neighbour on the right
   - 1.1 If not ascending order then swap
   - 1.2 Else continue with next two elements

   The last element is now the largest
2. Compare each element(except last 2) to its neighbour on the right
   - 2.1 If not ascending order then swap
   - 2.2 Else continue with next two elements

   The last 2 elements are now in the final positions
3. Repeat process till no more unsorted elements on left

Bubble sort example

13	8	17	2	63
8	13	17	2	63
8	13	2	17	63
8	13	2	17	63
8	13	2	17	63
8	2	13	17	63
8	2	13	17	63
8	2	13	17	63
2	8	13	17	63
2	8	13	17	63
2	8	13	17	63
2	8	13	17	63

LCSB

UNIVERSITÉ DU LUXEMBOURG

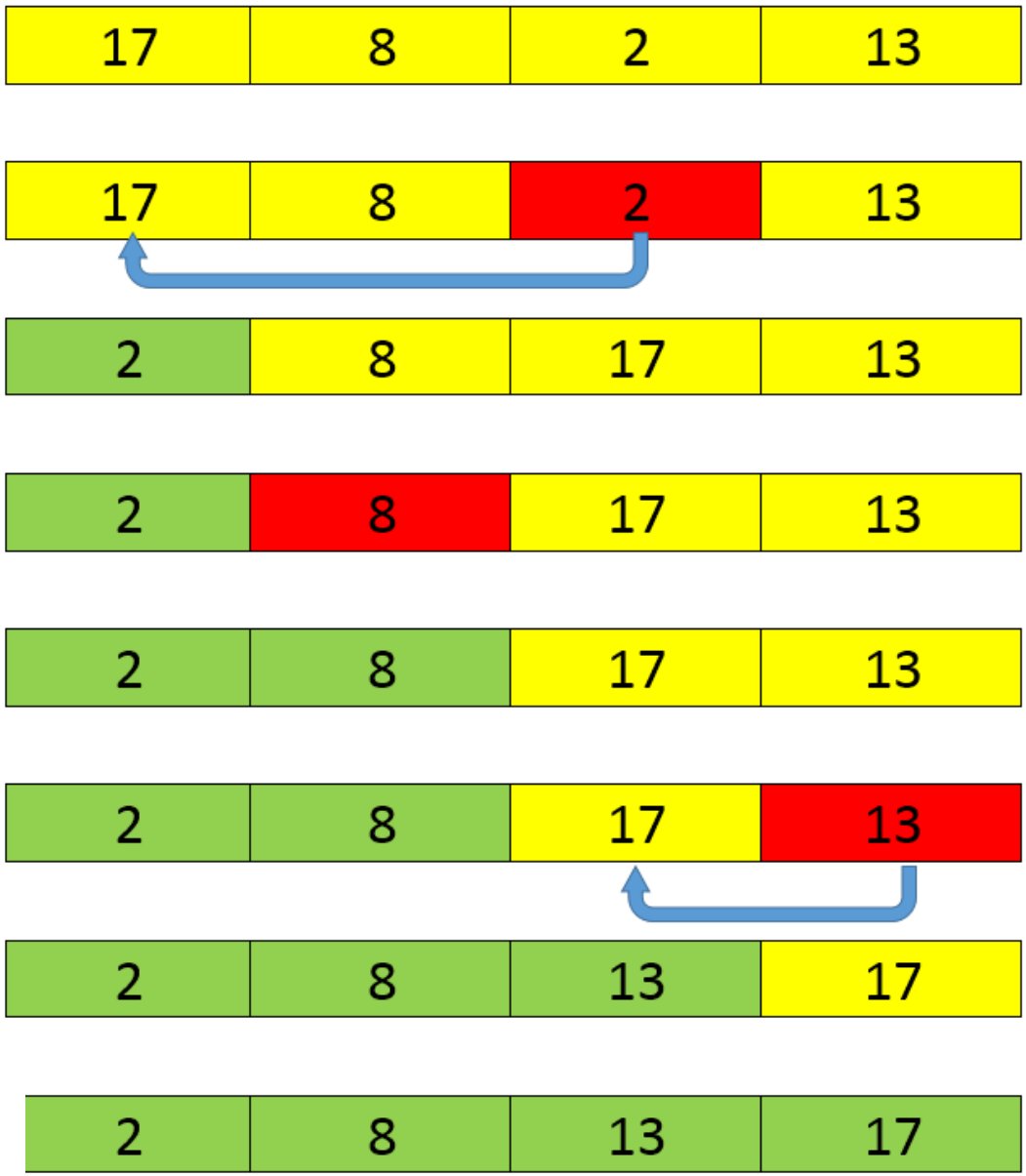
Selection sort

- *Select* the smallest element in the list and place in final position
- After each pass the smallest elements are sorted to the left of the list

Algorithm

1. Select the smallest element and swap with first position
2. Select the next smallest element and swap with second position
3. Repeat till no elements left

Selection sort example



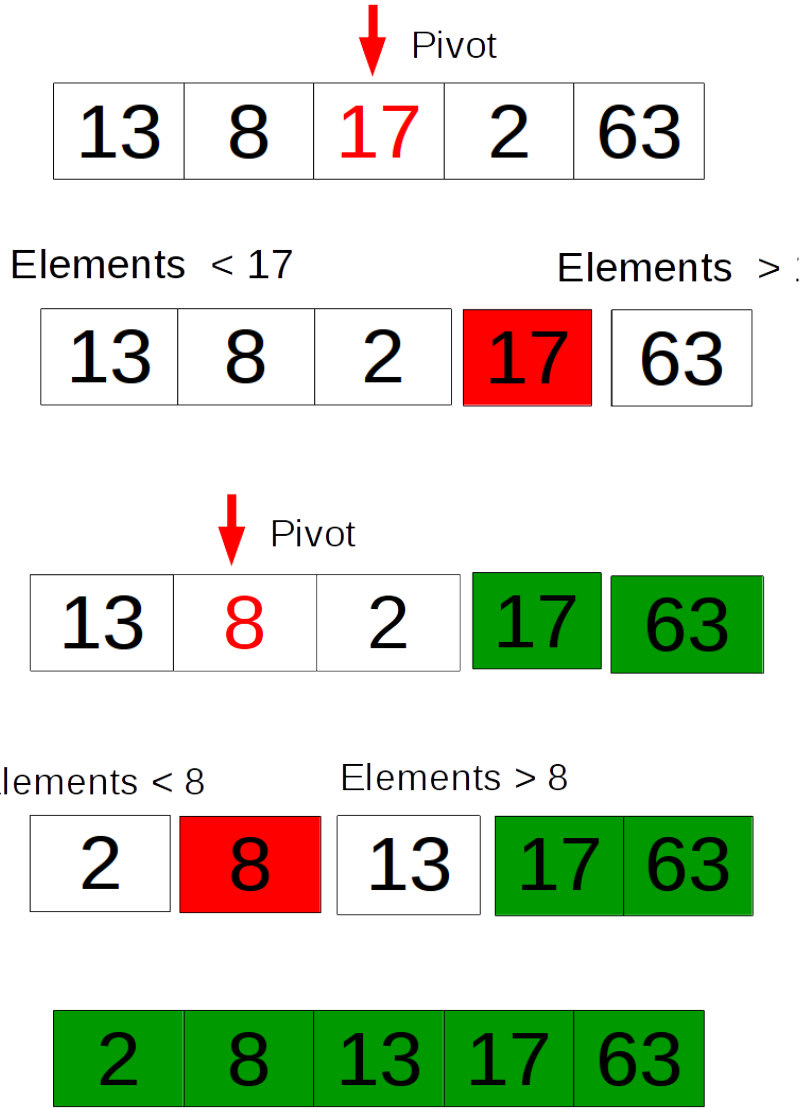
Quick Sort

- Example of a *divide and conquer* algorithm
- Select *pivot* element (random element). After each pass of the algorithm the pivot is in the final position
- Fast algorithm
- Recursive algorithm

Algorithm

1. Pick a pivot element
2. Place all elements smaller than the pivot to the left
3. Place all elements greater than the pivot to the right
   Now pivot is in final position
4. Repeat 1 to 3 for both lists *(recursion)*
   Divide and conquer
   - 4.1 Sublist on left of pivot (start to element before pivot)
   - 4.2 Sublist on right of pivot (element after pivot to last element)
5. Stop when no more elements in sublists

Example of Quick sort



Summary -Sorting Algorithms

1. Bubble Sort
   - - Comparisons at each step
   - + Stops when list is sorted
2. Selection Sort
   - - Sorted list also requires scanning all elements
   - + Requires only one swap per pass of algorithm
3. Quick Sort
   - - Selecting a pivot can be tricky
   - + Less comparisons and less swaps

Visualisation of Sorting

Some links given below to visualise the sorting stepwise for better understanding

- Comparison of sorting techniques
  `www.sorting-algorithms.com`
- Stepwise visualisation of sorting algorithms
  `www.comp.nus.edu.sg/~stevenha/visualization/sorting.html`
  `www.cs.usfca.edu/~galles/visualization/ComparisonSort.html`
- Youtube videos
  `https://www.youtube.com/user/AlgoRythmics/videos`

LCSB

UNIVERSITÉ DU LUXEMBOURG

Alignment

- What is alignment?
  Arrangement of two or more sequences to identify regions of similarity.
  Example use cases: Sequence similarity in proteins and DNA.
- Why alignment?
  Similarity in sequences helps to infer functional, structural or evolutionary relationships between them
- How to score an alignment?
  Distance(dissimilarity) between two strings.
  Example: Hamming distance and Levenshtein distance

Scoring alignments

- Hamming Distance

T	A	B	L	E	S
C	A	B	L	E	S

Hamming distance = Number of mismatches = 1

- Levenshtein or Edit distance

TABLES and RUMBLE

-	T	A	B	L	E	S
R	U	M	B	L	E	-
1	2	3				4

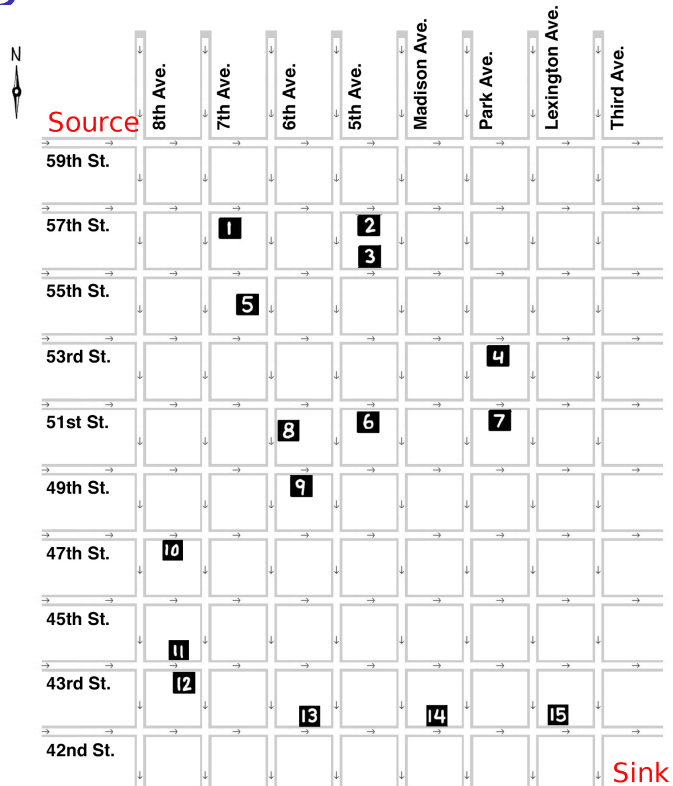
T	-	A	B	L	E	S
R	U	M	B	L	E	-
1	2	3				4

T	A	-	B	L	E	S
R	U	M	B	L	E	-
1	2	3				4

Allow substitution, insertion and deletion

Edit distance = Minimum number of edits required = 4

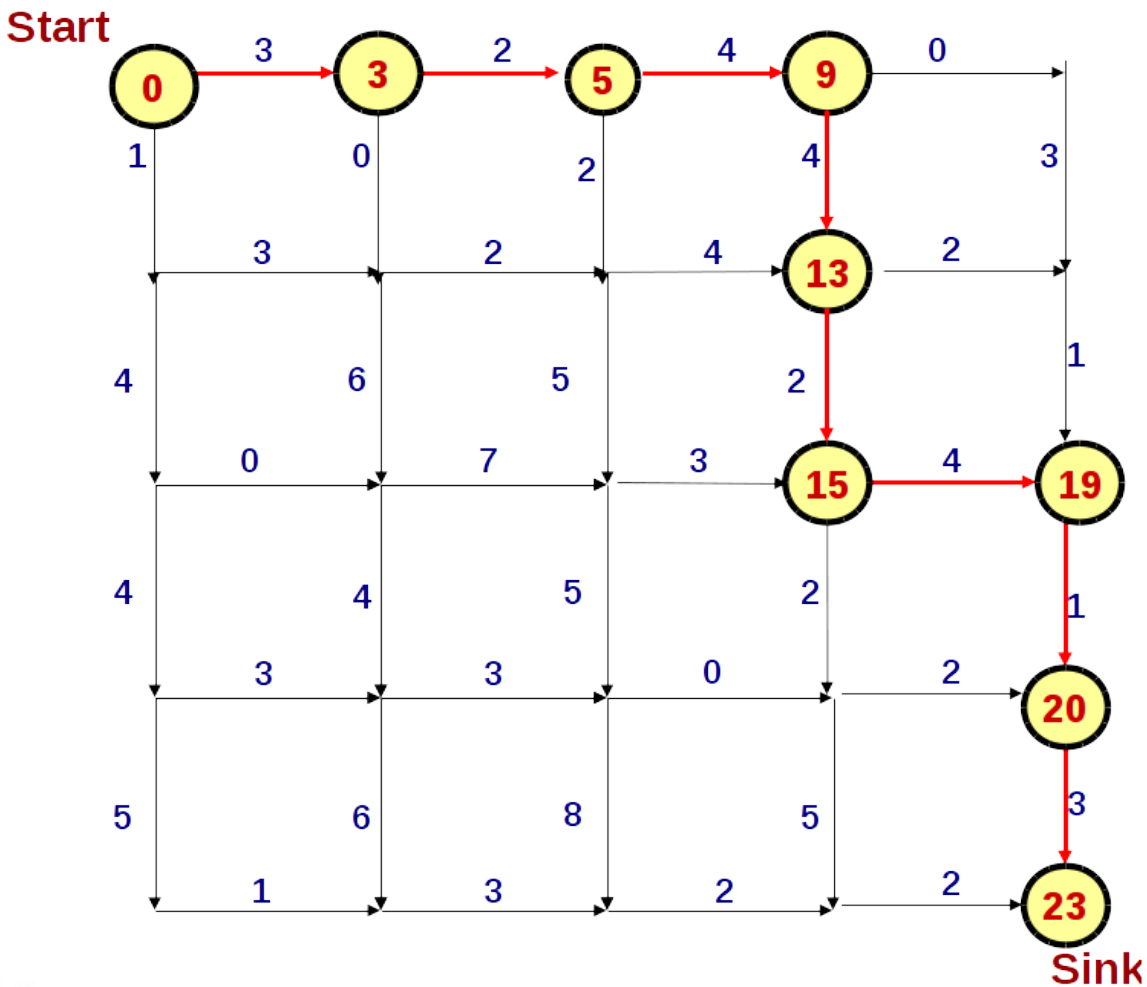
Alignment- Manhattan tourist problem



- Find the path with maximum number of tourist attractions.
- Longest (highest weighted) path from source to sink.
- Only possible movements **south** and **east**

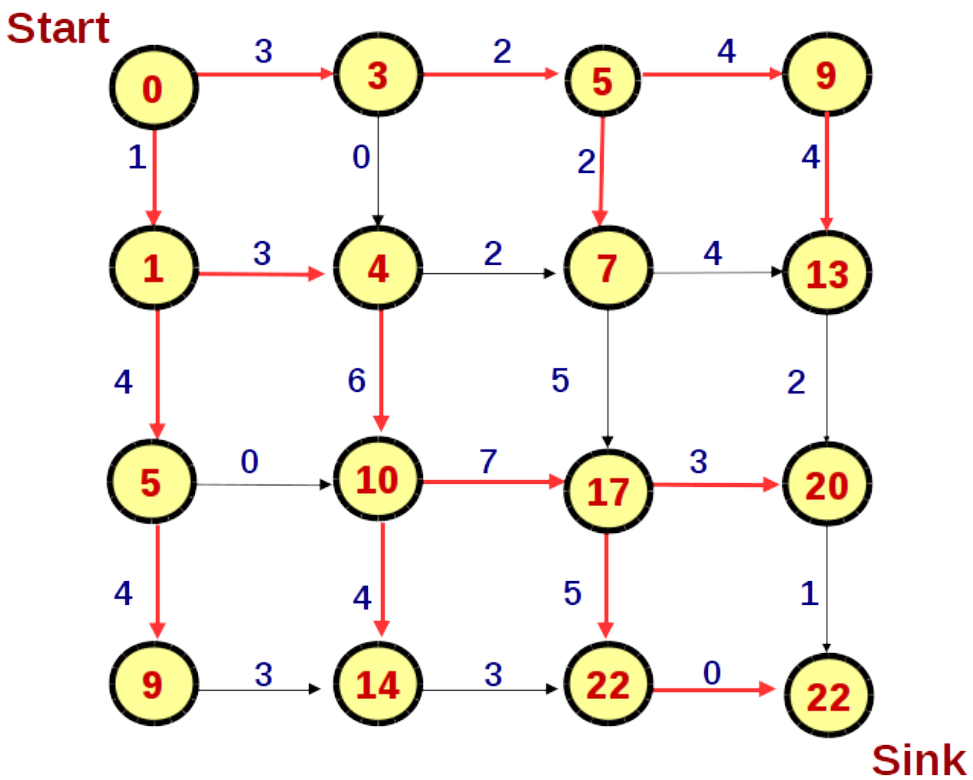
Source:Jones, N C., and Pavel Pevzner. An introduction to bioinformatics algorithms. MIT press, 2004.

MTP : Greedy approach

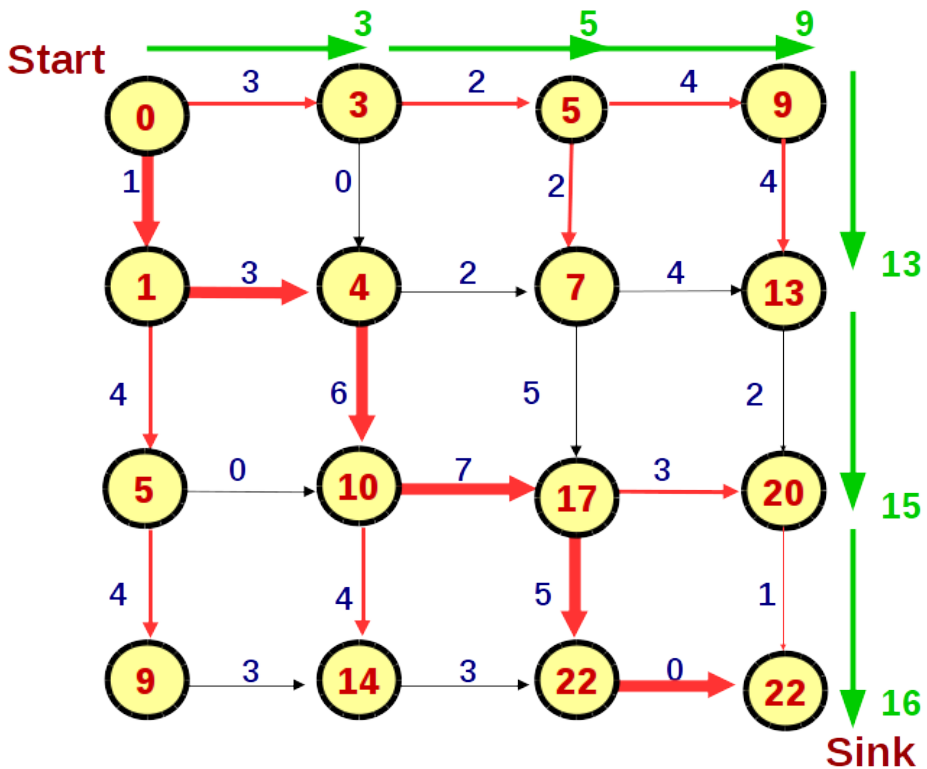


MTP : Score the nodes

$$NodeScore = Max\{(NodeScore_{North} + EdgeWeight_{North}), (NodeScore_{West} + EdgeWeight_{West})\}$$



MTP : Greedy vs Optimal



Needleman-Wunsch Algorithm (Global Alignment)

- Find best alignment over complete length of two sequences

Applications

- Comparing two genes with same function (in human vs. mouse).
- Comparing two proteins with similar function.

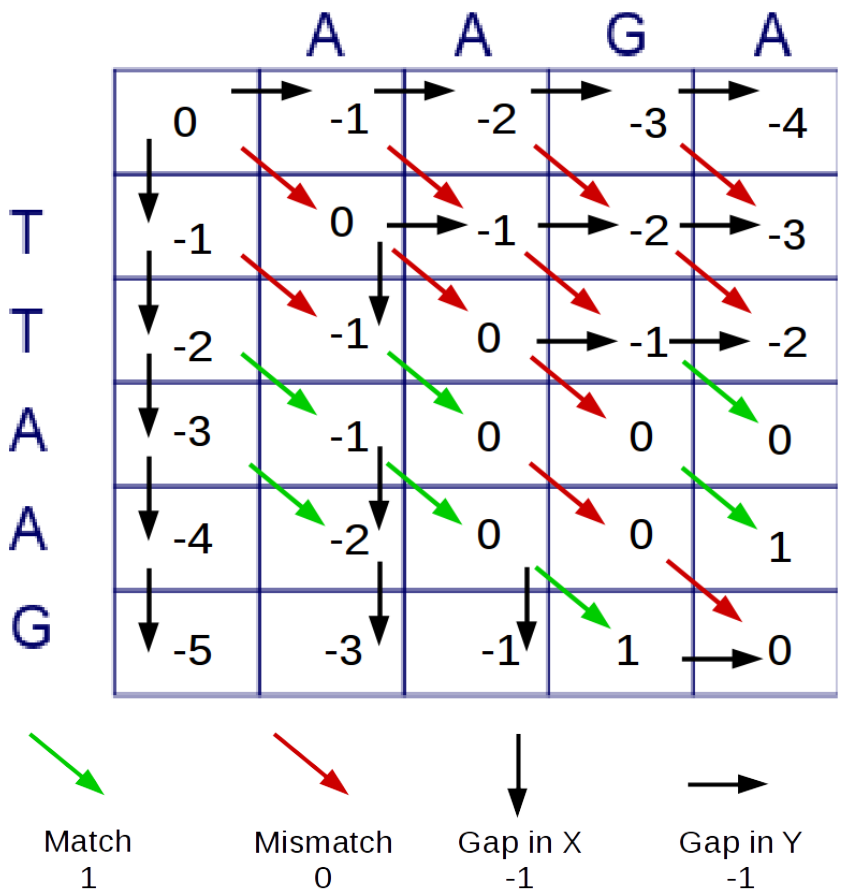
Steps

1. Initialisation of the score and traceback matrix
2. Calculation of scores and filling the trace
3. Traceback and find optimal alignment

Global Alignment : Score matrix and traceback matrix

Each node gets the trace from the node giving maximum score

$Score = max(Score_{NW} + Match/MisMatch, Score_N + gap, Score_w + gap)$



Backtracking for the alignment

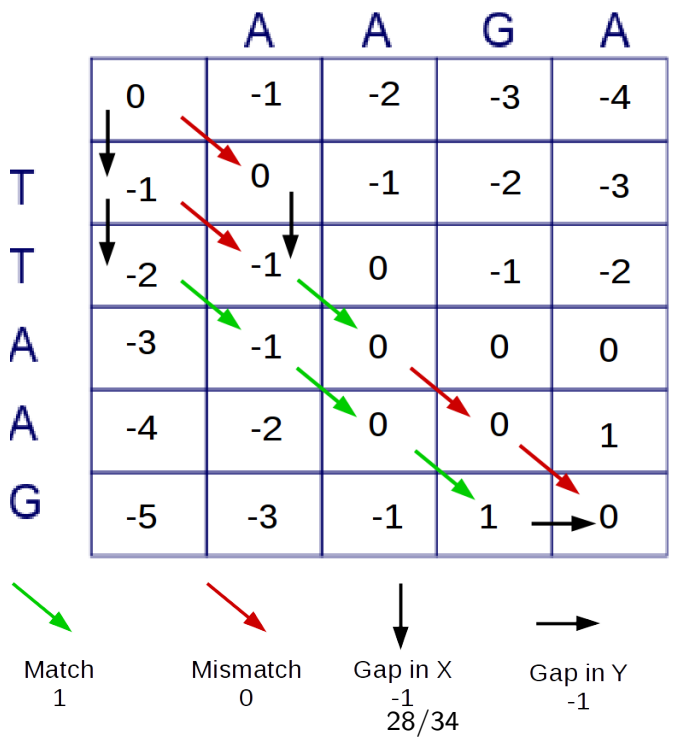
- Depending on the penalty for mismatch and gaps the optimal alignments will change
- Substitution matrices commonly used for Amino Acids PAM and BLOSUM

Optimal alignments

A	-	A	G	A
T	T	A	A	G

-	A	A	G	A
T	T	A	A	G

-	-	A	A	G	A
T	T	A	A	G	-



Smith-Waterman Algorithm (Local Alignment)

- Find short stretches of similarity between two sequences
- Best matches between subsequences of two sequences

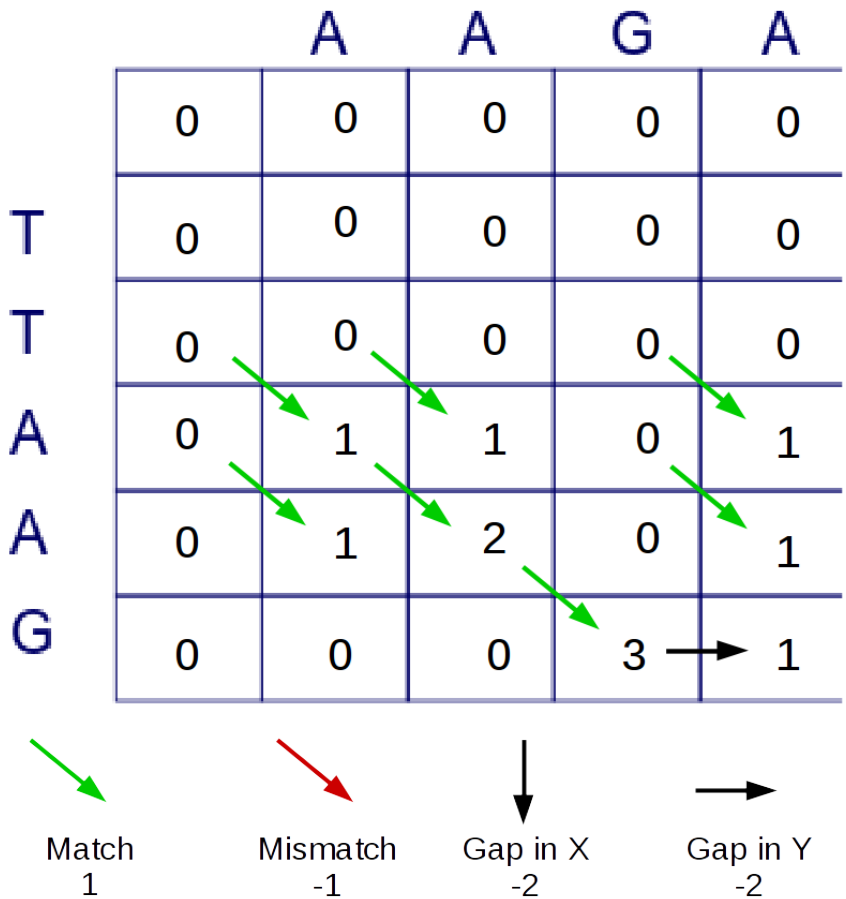
Application

- Comparing protein sequences that share a common motif or domain
- Comparing DNA sequences that share a similar motif

Local Alignment : Score matrix and traceback matrix

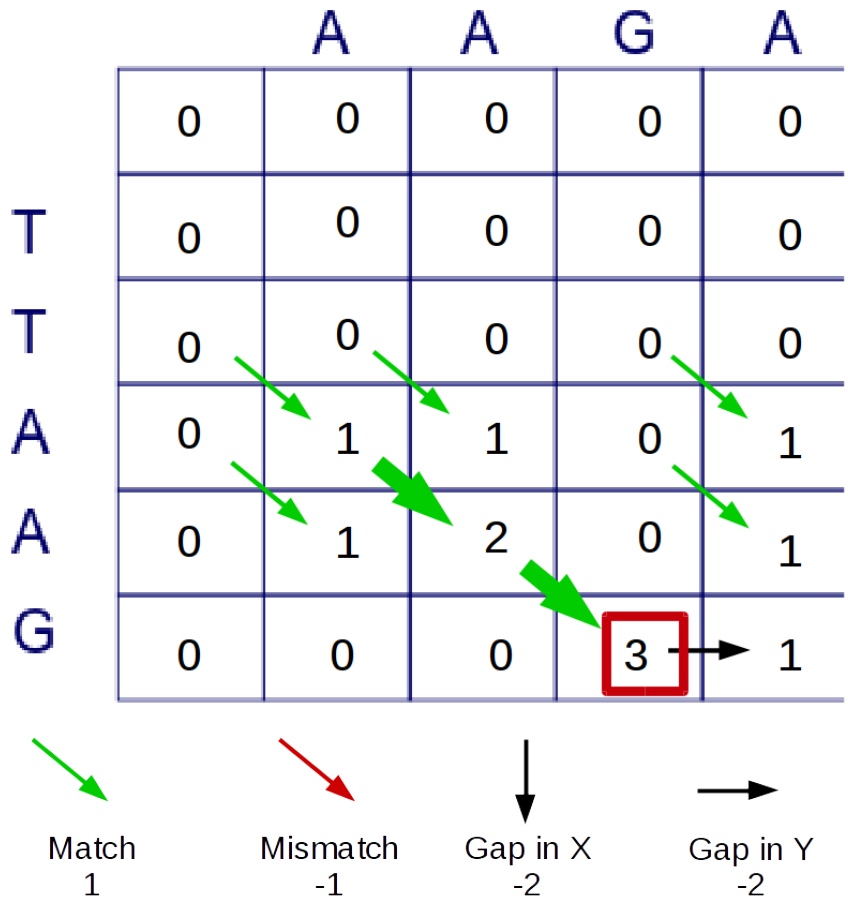
Each node gets the trace from the node giving maximum score

$Score = max(\mathbf{0}, Score_{NW} + Match/MisMatch, Score_N + gap, Score_w + gap)$



Local Alignment : backtracking

- Start from the highest node and traceback till a node with score 0
- The highest node need not be the last node, since the alignment is *local*



BLAST

- Basic Local Alignment Search Tool
- To search for nucleotide or protein sequences databases for similarity to a query sequence

Simplified blast steps

1. Setup - reads query sequence, parameters and the database to be searched. Prepares a set of short, fixed-length sequences based on the query (*seeds*)
2. Preliminary search - The database is scanned for matches (gap-free) with the words generated in step1. The matches with a certain score or above are used as seeds to score matches with gapped extensions
3. Traceback - Gapped seeds above a certain threshold are then used as seeds to score matches which also accounts for insertion, deletion and other parameters.

Some blast programs

- **BLASTP**: Compares an amino acid query sequence against a protein sequence database
- **BLASTN**: Compares a nucleotide query sequence against a nucleotide sequence database
- **BLASTX**: Searches a nucleotide query against a protein database, translating the query on the fly
- **TBLASTN**: Searches a protein query against a nucleotide database, translating the database on the fly.

Acknowledgement and references

- Jones, Neil C., and Pavel Pevzner. An introduction to bioinformatics algorithms. MIT press, 2004.
- Altschul, Stephen F., et al. "Basic local alignment search tool." Journal of molecular biology 215.3 (1990): 403-410.
- `www.bioalgorithms.info`
- `www.cs.umd.edu/class/fall2011/cmsc858s/Alignment.pdf`
- `www.ncbi.nlm.nih.gov/books/NBK153387/#BLAST.REF.2`

LCSB

UNIVERSITÉ DU LUXEMBOURG