Random Mutation Hill Climbing (RMHC)

Steps:

1. Choose a string at random. Call this string max_hilltop.
2. Choose a locus at random to flip. If the flip leads to an equal or higher fitness, then set max_hilltop to the resulting string.
3. Go to step 2 until an optimum string has been found or until a maximum number of evaluations have been performed.
4. Return the current value of max_hilltop.

Pseudocode:

```
function random_mutation_hill_climbing(max_number_of_iterations)
    max_hilltop <- generate_random()
    max_fitness <- compute_fitness(max_hilltop)
    for i in 0, max_number_of_iterations
        bit_position <- generate_random(1, length(max_hilltop))
        hilltop <- flip_bit(max_hilltop, bit_position)
        fitness <- compute_fitness(hilltop)
        if fitness >= max_fitness:
            max_fitness <- fitness
            max_hilltop <- hilltop
    return max_hilltop
```

Steepest Ascent Hill Climbing (SAHC)

Steps:

1. Choose a string at random. Call this string max-hilltop.
2. Going from left to right, systematically flip each bit in the string, one at a time, recording the fitnesses of the resulting one-bit mutants.
3. If any of the resulting one-bit mutants give a fitness increase, then set max-hilltop to the one-bit mutant giving the highest fitness increase.
4. If there is no fitness increase, then save max-hilltop and go to step 1. Otherwise, go to step 2 with the new max-hilltop.
5. When a set number of function evaluations has been performed, return the highest hilltop that was found.

Pseudocode:

```
function steepest_ascent_hill_climbing(max_number_of_iterations)
    max_hilltops <- {}
    for i in 0, max_number_of_iterations
        max_hilltop <- generate_random()
        max_fitness <- compute_fitness(max_hilltop)
        increase_found <- true
        while increase_found
            increase_found <- false
            for bit_position in 1, length(max_hilltop)
                hilltop <- flip_bit(max_hilltop, bit_position)
                fitness <- compute_fitness(hilltop)
                if fitness > max_fitness:
                    max_fitness <- fitness
                    max_hilltop <- hilltop
                    increase_found <- true
        max_hilltops <- max_hilltops U max_hilltop
    return max(max_hilltops)
```

Next Ascent Hill Climbing NAHC

Steps:

1. Choose a string at random. Call this string max-hilltop.
2. For i from 1 to l (where l is the length of the string), flip bit i; if this results in a fitness increase, keep the new string, otherwise flip bit i; back. As soon as a fitness increase is found, set max-hilltop to that increased fitness string without evaluating any more bit flips of the original string. Go to step 2 with the new max-hilltop, but continue mutating the new string starting immediately after the bit position at which the previous fitness increase was found.
3. If no increases in fitness were found, save max-hilltop and go to step 1.
4. When a set number of function evaluations has been performed, return the highest hilltop that was found.

Pseudocode:

```
function next_ascent_hill_climbing(max_number_of_iterations)

    max_hilltops <- {}

    for i in 0, max_number_of_iterations

        max_hilltop <- generate_random()

        max_fitness <- compute_fitness(max_hilltop)

        for bit_position in 1, length(max_hilltop)

            hilltop <- flip_bit(max_hilltop, bit_position)

            fitness <- compute_fitness(hilltop)

             if fitness > max_fitness:

                max_fitness <- fitness

                max_hilltop <- hilltop

        max_hilltops <- max_hilltops U max_hilltop

    return max(max_hilltops)
```