Prosjekt 2, C++ data structures

Problemstilling

- Målet her er å implementere noen enkle data strukturer, utvide disse og se på hvordan de oppfører seg mhp effektivitet.
- Oppgaven er delt i 4 deler der vi ser på forskjellige typer arrays og lister i hver del.
- Part 1: Dynamic Arrays
- Part 2: Linked lists
- Part 3: Comparing the ArrayList and the LinkedList
- Part 4: Circularly Linked List

Metode og utfordringer

Part 2, vi implementerer flere metoder slik som i part 1; print, append, remove.. etc

Part 1 Dynamic arrays

```
//Here we create the class ArrayList
class ArrayList {
    private:
    int *data;
    int growth = 2;
    int capacity = 1;
    public:
      int size = 0;
      ArrayList(){
        data = new int[capacity];
    }

    ArrayList( vector<int> initial) {
      size = 0;
      capacity = 1024;
      data = new int[capacity];

      for (int e: initial) {
          append(e);
      }
  }
```

Legger til forskjellige metoder, noen public og noen private.

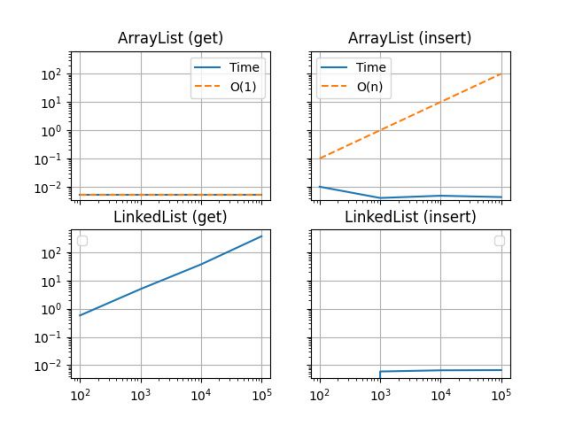
I part 2 skal vi implementere en lenket liste. Forskjellen mellom en dynamic arraylist og linked list er at i en lenket liste så er hvert element lagret i en egen struktur som vi kaller en node.

Noden lagrer sin egen verdi i tillegg peker den til neste node, derav “lenket” liste.

```
Node{
  int value;
  Node *next = nullptr;
  Node *prev = nullptr;
  Node(int value) : value(value)
  {
  }
  Node(int value, Node *next) : value(value), next(next)
  {

  }
};


class LinkedList
{
private:
  Node *head = nullptr;
  Node *tail = nullptr;
public:
  LinkedList(){

  }
  void append(int value)
  {
    Node *node = new Node{value};
    if (head==nullptr){
      head = node;
      return;
    }
    Node *current;
    current = head;
    while (current->next){
      current = current->next;
    }
    current->next=node;
  }
```

Metode og utfordringer

- Et av målene med dette prosjektet er at i part 3 så skal vi sammenligne de to typene datastruktur vi har laget og utvidet med metoder. Vi gjør denne sammenligningen med koden gitt i oppgaven og skriver output til to forskjellige .txt filer som vi tar inn i en python kode for å plotte og se på performance. Det som er vist i plottet under er lang tid det tar å hente N/2 elementet altså elementet i midten av listen når vi har fylt den med N elementer.



Part 3 Circular linked list

```
struct
Node{
  int value;
  Node *next=nullptr;
  Node(int value) : value(value)
  {
  }
  Node(int value, Node *next) : value(value), next(next)
  {

  }
};


class CircLinkedList
{
private:
  Node *head = nullptr;
public:
  CircLinkedList(){

  }
  void append(int value)
  {
    Node *node = new Node{value};
    if (head==nullptr){
      head = node;
      node -> next=head;
      return;
    }
    Node *current;
    current = head;
    while (current->next != head){
      current = current->next;
    }
    current->next=node;
    node -> next = head;
  }

```

Fungerer det? Testing

Tester for primtall (part 1)

```
bool is_prime(int n) {
    if (n == 1) {
        return false;
    }

    for (int d=2; d<n; d++) {
        if (n % d == 0) {
            return false;
        }
    }
    return true;
}


//test funksjon utenfor classen
void test_is_prime(){
    int n=1;
    ArrayList a;
    while(a.length() <10){
        if (is_prime(n)){
            a.append(n);
        }
        n++;
    }
    a.print();
};
```

```
172
173  int main(){
174    LinkedList ll{};
175    ll.append(2);
176    ll.append(3);
177    ll.append(1);
178    ll.print();
179
180    ll.insert(77,2);
181    ll.print();
182    cout<< ll.pop(2) <<endl;
183    ll.remove(2);
184    ll.print();
185    ll.pop();
186    ll.print();
187    return  0;
188  }
```