PmergeMe

Objetivo del programa:

Debes crear un programa llamado **PmergeMe** que, al recibir como argumentos una secuencia de enteros positivos, los ordene utilizando el algoritmo de *merge–insert sort* (también conocido como algoritmo Ford–Johnson, según lo descrito en "The Art Of Computer Programming, Vol.3. Merge Insertion, Page 184").

Requisitos clave del ejercicio:

- **Entrada:**
  - El programa se ejecuta pasando como argumentos de línea de comandos una secuencia de números enteros positivos.
  - Cada número debe ser validado; si se encuentra algún valor negativo o no numérico, el programa debe mostrar un mensaje de error en la salida estándar de error.
- **Algoritmo de ordenamiento:**
  - Se debe implementar el algoritmo *merge–insert sort* (Ford–Johnson). Este algoritmo combina ideas de *merge sort* (ordenamiento por fusión) e *insertion sort* (ordenamiento por inserción) y se diseña para minimizar el número de comparaciones.
  - Específicamente, el algoritmo divide la secuencia en pares, ordena cada par, y luego fusiona las subsecuencias mediante inserciones binarias. La inserción de elementos se realiza siguiendo un orden especial basado en los números de Jacobsthal, lo que permite optimizar el proceso de búsqueda binaria.
- **Uso de contenedores:**
  - Debes utilizar **al menos dos contenedores diferentes** de la STL (por ejemplo, std::vector y std::deque o std::list) para almacenar la secuencia.
  - Se recomienda implementar el algoritmo por separado para cada contenedor (es decir, no se debe escribir una función genérica para ambos) para poder comparar sus rendimientos.
- **Rendimiento y salida:**
  - El programa debe ser capaz de manejar al menos 3000 números.
  - La salida en la consola deberá mostrar:
    - Una línea con un texto explícito seguido de la secuencia de enteros desordenada.
    - Una línea con otro texto explícito seguido de la secuencia ordenada.
    - Una línea que indique el tiempo empleado en procesar el ordenamiento utilizando el primer contenedor.
    - Otra línea que indique el tiempo empleado para el segundo contenedor.
  - El tiempo de procesamiento debe medirse con suficiente precisión (por ejemplo, usando std::clock() junto con std::fixed y std::setprecision) para poder apreciar las diferencias entre contenedores.

2. Propósito de aprendizaje

Este ejercicio tiene varios objetivos pedagógicos muy importantes:

- **Uso de la STL y contenedores:**

  Aunque el ejercicio podría resolverse sin ellos, el objetivo es **practicar el uso de la Standard Template Library (STL)** de C++98. Se espera que apliques contenedores estándar (como vector, deque o list) y algoritmos (por ejemplo, std::lower_bound para la búsqueda binaria).

- **Comprender algoritmos de ordenamiento avanzados:**

  El algoritmo merge–insert (Ford–Johnson) es especialmente interesante porque está diseñado para minimizar el número de comparaciones. Esto te obliga a analizar en detalle la mecánica de los algoritmos híbridos que combinan técnicas de fusión e inserción, así como a entender el papel de la secuencia de Jacobsthal para optimizar la búsqueda binaria.

- **Medición y análisis de rendimiento:**

  Deberás aprender a medir tiempos de ejecución en C++98, lo cual es fundamental para realizar benchmarking y comparar el rendimiento de diferentes contenedores en función de cómo gestionan la memoria y la eficiencia en las operaciones.

- **Manejo de errores y validación de entrada:**

  Es fundamental implementar una robusta validación de datos: si se reciben valores negativos o argumentos mal formateados, el programa debe gestionar el error mostrando mensajes apropiados.

- **Organización modular del código:**

  El ejercicio te invita a dividir la solución en distintos módulos y ficheros (por ejemplo, separando la lógica del algoritmo en archivos .cpp y sus correspondientes .hpp). Además, se permite el uso de templates en los headers, lo que fomenta buenas prácticas de organización y mantenimiento del código.

3. Cómo afrontarlo en C++98

a) Procesamiento de argumentos y validación

- **Recepción de argumentos:**

  Utiliza la función main(int argc, char *argv[]) para capturar la secuencia de enteros.

- **Validación:**

Para cada argumento, emplea std::istringstream para convertir la cadena a entero y verifica que:

- Sea un entero válido.
- Sea un número positivo.
- (Opcional) Cumpla con algún rango establecido, si es necesario.

- **Gestión de errores:**

  Si algún argumento es inválido, muestra un mensaje de error (por ejemplo, "Error") y termina la ejecución.

b) Uso de dos contenedores de la STL

- **Selección de contenedores:**

  Dado que en ejercicios anteriores se han utilizado algunos contenedores (por ejemplo, std::vector o std::map), en este ejercicio se deben usar al menos dos contenedores diferentes. Una estrategia común es usar, por ejemplo:

  - std::vector<int> para una implementación.
  - std::deque<int> o std::list<int> para la otra.

- **Almacenamiento de la secuencia:**
  Al recibir la secuencia, almacena los números en ambos contenedores para poder aplicar el algoritmo por separado y comparar su rendimiento.

c) Implementación del algoritmo merge–insert (Ford–Johnson)

- **División en pares y ordenación interna:**

  Divide la secuencia en pares. Para cada par, compara los dos elementos y ordénalos de forma que el menor quede a la izquierda y el mayor a la derecha.

- **Construcción de la cadena principal ("main chain") y la "pend":**

  Extrae los elementos mayores de cada par para formar la secuencia principal, la cual estará (idealmente) ordenada. Los elementos menores (los “pend”) se almacenarán en otra secuencia.

- **Inserción binaria optimizada:**

  Inserta los elementos de la “pend” en la cadena principal utilizando búsqueda binaria. El orden de inserción debe seguir la secuencia especial determinada por los números de Jacobsthal, lo que minimiza el número de comparaciones.

- **Implementación independiente para cada contenedor:**

Es importante implementar el algoritmo de forma separada para cada contenedor, ya que la forma de insertar y recorrer puede variar (por ejemplo, la eficiencia de std::vector vs. std::deque).

d) Medición de tiempos de ejecución

- **Uso de std::clock():**

  En C++98, la forma habitual de medir el tiempo es utilizando la función std::clock() del header <ctime>.

  - Toma el tiempo antes de iniciar el algoritmo para cada contenedor.
  - Toma el tiempo inmediatamente después de finalizar el proceso.
  - Calcula la diferencia y muestra el tiempo en microsegundos o la unidad que se elija, utilizando std::fixed y std::setprecision para dar la precisión necesaria.

- **Comparación:**

  Imprime en la salida el tiempo empleado para ordenar la secuencia con cada contenedor, permitiendo ver diferencias en rendimiento.

e) Impresión de resultados

- **Formato de salida:**
  - Primera línea: Un texto explícito seguido de la secuencia original (sin ordenar).
  - Segunda línea: Un texto explícito seguido de la secuencia ordenada.
  - Tercera y cuarta línea: Mensajes que indiquen el tiempo de procesamiento para cada contenedor, con la precisión suficiente para notar la diferencia.

4. Recursos en video recomendados (en español)

Para profundizar en los conceptos necesarios y abordar con éxito el ejercicio, se recomiendan los siguientes videos en español (asegúrate de que estén disponibles actualmente):

1. **Merge Sort C++ | Ordenamiento por Mezcla – YouTube**

   Descripción: Explicación paso a paso del algoritmo Merge Sort en C++, útil para comprender la estrategia "divide y vencerás" que se utiliza también en el merge–insert sort.

   Enlace: https://www.youtube.com/watch?v=1Xl-Zavaci8

2. **Curso de C++ Avanzado. STL. Algoritmos de la STL – Píldoras Informáticas**

   Descripción: Este video profundiza en el uso de algoritmos de la STL y explica conceptos avanzados que son muy útiles al implementar algoritmos de ordenamiento optimizados.

Enlace: https://www.youtube.com/watch?v=IebP5c4MABw

Resumen

El ejercicio **PmergeMe** consiste en desarrollar un programa en C++98 que ordene una secuencia de enteros positivos utilizando el algoritmo merge–insert (Ford–Johnson), con la obligación de emplear al menos dos contenedores diferentes de la STL y medir el tiempo de ejecución para cada uno. Su propósito es profundizar en el uso de contenedores, algoritmos híbridos para minimizar comparaciones, manejo de errores, medición de rendimiento y una correcta organización modular del código en C++98.

La clave para abordarlo es:

- Validar y procesar correctamente la entrada.
- Almacenar la secuencia en dos contenedores distintos.
- Implementar el algoritmo merge–insert de forma separada para cada contenedor, prestando atención a la estrategia de inserción basada en números de Jacobsthal.
- Medir y comparar los tiempos de ejecución para apreciar las diferencias de rendimiento entre contenedores.

Con estos conceptos y recursos, estarás bien preparado para desarrollar una solución óptima y cumplir con los objetivos del ejercicio.

El código

A continuación se detalla de forma clara y completa qué hace y cómo funciona el código, dividiéndolo en secciones para facilitar su comprensión.

1. Objetivo General

El código implementa la clase **PmergeMe**, cuyo propósito es ordenar una secuencia de números enteros positivos recibidos como argumentos de línea de comandos. Se utiliza una variante del algoritmo *merge sort* (ordenamiento por fusión) implementado de forma recursiva, pero aplicado de dos maneras: una versión que utiliza un **std::vector** y otra que utiliza una **std::list**. Además, se mide el tiempo de ejecución (en microsegundos) de cada implementación para comparar el rendimiento entre contenedores.

2. Estructura General del Código

Archivos

- **PmergeMe.hpp:**

- Define la clase PmergeMe y declara sus métodos públicos:

    - sortList(int argc, char *argv[]) para ordenar utilizando una lista.
    - sortVector(int argc, char *argv[]) para ordenar utilizando un vector.

También se declara una excepción interna InvalidInput que se lanza cuando la entrada es inválida.

- **PmergeMe.cpp:**

Contiene la implementación de los métodos de la clase PmergeMe y funciones auxiliares para:

- Convertir cadenas a enteros sin signo.
- Imprimir parcialmente (los primeros 7 elementos) un vector o una lista.
- Realizar la unión (merge) de dos subestructuras ordenadas (funciones joinVec y joinList).
- Implementar de forma recursiva el algoritmo de ordenamiento (*merge sort*) para vector (insertVec) y para lista (insertList).

- **main.cpp:**

Es el punto de entrada del programa. Realiza lo siguiente:

- Verifica que se hayan pasado argumentos (más allá del nombre del ejecutable).
- Valida que cada argumento contenga únicamente dígitos y que no existan elementos duplicados.
- Crea un objeto PmergeMe y llama a los métodos sortVector y sortList para ordenar la secuencia de números, mostrando antes y después el resultado junto con el tiempo de ejecución.

3. Detalle del Funcionamiento

a) Manejo de la Entrada y Validación

- **Conversión de cadena a entero:**

  La función ft_stou (definida en main.cpp) utiliza un std::stringstream para convertir una cadena a un unsigned int.

- **Validación de argumentos:**

En main.cpp se comprueba que:

  - Se haya proporcionado al menos un número (argc < 2 produce error).
  - Cada argumento contenga solo dígitos (se usa find_first_not_of("0123456789")).
  - No existan números duplicados (se hace una comparación doble entre elementos del vector temporal).

- **Excepción personalizada:**

  Si alguna de las comprobaciones falla, se lanza la excepción PmergeMe::InvalidInput, la cual sobreescribe el método what() para devolver el mensaje "Invalid input" (mostrado en rojo mediante macros definidas).

b) Ordenamiento con Vector (sortVector)

- **Carga de datos:**

  Se recorre argv (a partir de índice 1) y cada argumento se convierte a unsigned int usando ft_stou y se almacena en un std::vector<unsigned int>.

- **Impresión inicial:**

  La función printVector recorre el vector e imprime los primeros 7 elementos (y si hay más, imprime "[...]") para dar una vista previa de la secuencia desordenada.

- **Ordenamiento Recursivo:**

  La función insertVec implementa un **merge sort** de forma recursiva:

  - Si el vector tiene 1 o menos elementos, ya está ordenado.
  - Divide el vector en dos mitades: la izquierda (id) y la derecha (dd).
  - Llama recursivamente a insertVec sobre cada mitad para ordenarlas.
  - Luego, se unen ambas mitades ordenadas utilizando la función joinVec, que realiza el merge clásico:
    - Mientras ambos vectores tengan elementos, se compara el primer elemento (front) de cada uno y se va empujando el menor al vector resultado.
    - Cuando uno se vacía, se añaden los elementos restantes del otro.

- **Medición de tiempo:**

Se usa std::clock() para tomar el tiempo antes y después del ordenamiento. La diferencia se convierte a microsegundos (usando la macro MICROSEC) y se imprime junto al número de elementos ordenados.

- **Impresión final:**

  Se muestra el vector ordenado (nuevamente solo los primeros 7 elementos) y el tiempo de ejecución.

c) Ordenamiento con Lista (sortList)

- **Carga de datos:**

  Similar al vector, se recorre argv y se insertan los números convertidos en un std::list<unsigned int>.

- **Impresión inicial:**

  La función printList recorre la lista e imprime los primeros 7 elementos, mostrando "[...]" si hay más.

- **Ordenamiento Recursivo:**

  La función insertList realiza un **merge sort** adaptado para listas:

  - Se divide la lista en dos partes:
    - Se extraen los primeros *mid* elementos para formar la sublista izquierda.
    - El resto se asigna a la sublista derecha.
  - Se aplican llamadas recursivas a insertList en cada sublista.
  - Se unen las dos sublistas ordenadas mediante la función joinList, la cual es similar a joinVec pero opera sobre std::list:
    - Se compara el primer elemento (front) de cada lista, se añade el menor al resultado y se elimina de la lista original.
    - Se agregan posteriormente los elementos restantes de cada sublista.
- **Medición de tiempo:**

  De igual forma, se utiliza std::clock() para medir el tiempo total de ordenamiento de la lista y se muestra en microsegundos.

- **Impresión final:**

  Se imprime la lista ordenada y el tiempo de ejecución.

d) Uso de Contenedores y Funciones Estáticas

- **Contenedores:**

  Se utilizan dos contenedores de la STL: std::vector<unsigned int> y std::list<unsigned int>. Esto permite comparar el rendimiento de ambos en el mismo algoritmo de ordenamiento.

- **Funciones estáticas auxiliares:**
  - printVector y printList se utilizan para mostrar parcialmente el contenido de cada contenedor.
  - joinVec y joinList realizan la fusión de dos secuencias ordenadas.
  - insertVec y insertList implementan la parte recursiva del merge sort para cada contenedor.
- **Medición de tiempo:**

  La macro MICROSEC define la cantidad de microsegundos por segundo (1.000.000), para convertir el tiempo devuelto por std::clock() a microsegundos.

4. Flujo del Programa en main.cpp

1. **Verificación de argumentos:**
   a. Si no se pasan argumentos, se imprime un mensaje de error mediante la excepción InvalidInput.
2. **Validación de cada argumento:**
   a. Se comprueba que cada argumento contenga solo dígitos.
   b. Se almacenan temporalmente en un vector para luego comprobar que no existan duplicados.
3. **Instanciación de PmergeMe:**
   a. Se crea un objeto PmergeMe.
4. **Ordenamiento y Medición:**
   a. Se llama a sortVector(argc, argv) para ordenar la secuencia usando un vector.
   b. Se imprime una línea en blanco y se llama a sortList(argc, argv) para ordenar la misma secuencia usando una lista.
5. **Finalización:**
   a. Si todo ha ido bien, el programa retorna EXIT_SUCCESS.

Resumen Final

El código de **PmergeMe** se encarga de:

- **Recibir y validar** una secuencia de números enteros positivos como argumentos.
- **Convertir** dichos argumentos a unsigned int y almacenarlos en dos contenedores distintos (vector y lista).
- **Ordenar** cada contenedor utilizando un algoritmo recursivo de **merge sort** (dividir en dos partes, ordenar cada parte y fusionarlas).
- **Medir el tiempo de ejecución** del ordenamiento para cada contenedor y mostrar los resultados junto con una vista previa (los primeros 7 elementos) de la secuencia antes y después de ordenar.
- **Manejar errores** mediante una excepción personalizada que se lanza si se detecta entrada no válida (caracteres no numéricos o duplicados).

Este código es un ejemplo práctico de cómo aplicar conceptos de algoritmos recursivos, manejo de contenedores de la STL, validación de datos y medición de rendimiento en C++98.

Esta explicación resume de forma detallada y sencilla el funcionamiento del código, desde la validación de la entrada hasta la ejecución y medición del algoritmo de ordenamiento en ambos contenedores.