## *PROBLEM STATEMENT*

**Compare Cheney's algorithm and Mark-compact algorithm with C++ and demonstrate which is better and why?. A C++ Code demonstrating the principles of both this algorithm is expected. Documentation with comparison of both these algorithms including the Code, Graphs, and explanation is expected.**

## OBJECTIVE :

**TO Compare Cheney's algorithm and Mark-compact algorithm with C++ and to find out which is better and why ? with code, gragh and explanation.**

**From the given problem we come to know that ,the algorithms are based on garbage collections.**

## WHAT IS GARBAGE COLLECTIONS ?

**garbage collection (GC) is a form of automatic memory management. The garbage collector, or just collector, attempts to reclaim garbage, or memory occupied by objects that are no longer in use by the program. Garbage collection was invented by American computer scientist John McCarthy around 1959 to simplify manual memory management in Lisp.**

## ADVANTAGES OF GARBAGE COLLECTION :

**Garbage collection frees the programmer from manually dealing with memory deallocation. As a result, certain categories of bugs are eliminated or substantially reduced:**

**Dangling pointer bugs, which occur when a piece of memory is freed while there are still pointers to it, and one of those pointers is dereferenced. By then the memory may have been reassigned to another use, with unpredictable results.**
**Double free bugs, which occur when the program tries to free a region of memory that has already been freed, and perhaps already been allocated again.**
**Certain kinds of memory leaks, in which a program fails to free memory occupied by objects that have become unreachable, which can lead to memory exhaustion. (Garbage collection typically[who?] does not deal with the unbounded**

**accumulation of data that is reachable, but that will actually not be used by the program.)**
**Efficient implementations of persistent data structures**
**Some of the bugs addressed by garbage collection have security implications.**

**DISADVANTAGES OF GARBAGE COLLECTION?**

**Typically, garbage collection has certain disadvantages, including consuming additional resources, performance impacts, possible stalls in program execution, and incompatibility with manual resource management.**

**Garbage collection consumes computing resources in deciding which memory to free, even though the programmer may have already known this information. The penalty for the convenience of not annotating object lifetime manually in the source code is overhead, which can lead to decreased or uneven performance.A peer-reviewed paper from 2005 came to the conclusion that GC needs five times the memory to compensate for this overhead and to perform as fast as explicit memory management.Interaction with memory hierarchy effects can make this overhead intolerable in circumstances that are hard to predict or to detect in routine testing. The impact on performance was also given by Apple as a reason for not adopting garbage collection in iOS despite being the most desired feature.**

**The moment when the garbage is actually collected can be unpredictable, resulting in stalls (pauses to shift/free memory) scattered throughout a session. Unpredictable stalls can be unacceptable in real-time environments, in transaction processing, or in interactive programs. Incremental, concurrent, and real-time garbage collectors address these problems, with varying trade-offs.**

## What is Garbage Collection?

- *What is Garbage Collection?*
  - Finding garbage and reclaiming memory allocated to it.
- *Why Garbage Collection?*
  - the heap space occupied by an un-referenced object can be recycled and made available for subsequent new objects
- *When is the Garbage Collection process invoked?*
  - When the total memory allocated to a Java program exceeds some threshold.
- *Is a running program affected by garbage collection?*
  - Yes, the program suspends during garbage collection.

**METHEDOLOGY :**

- **ALGORITHM USED :**

**~ CHENEY'S ALGORITHM**

**~MARK COMPACT. ALGORITHM**

(1) **CHENEY'S ALGORITHM**

**Cheney's Algorithm uses a very simple allocation algorithm and has no need for stack (since is not recursive). More importantly, its run time depends on the number of live objects, not on the heap size. Furthermore, it does not suffer from data fragmentation, resulting into a more compact memory. In addition, it allows incremental (concurrent) garbage collection. On the other hand, it needs double the amount of memory and needs to recognize all pointers to heap.**

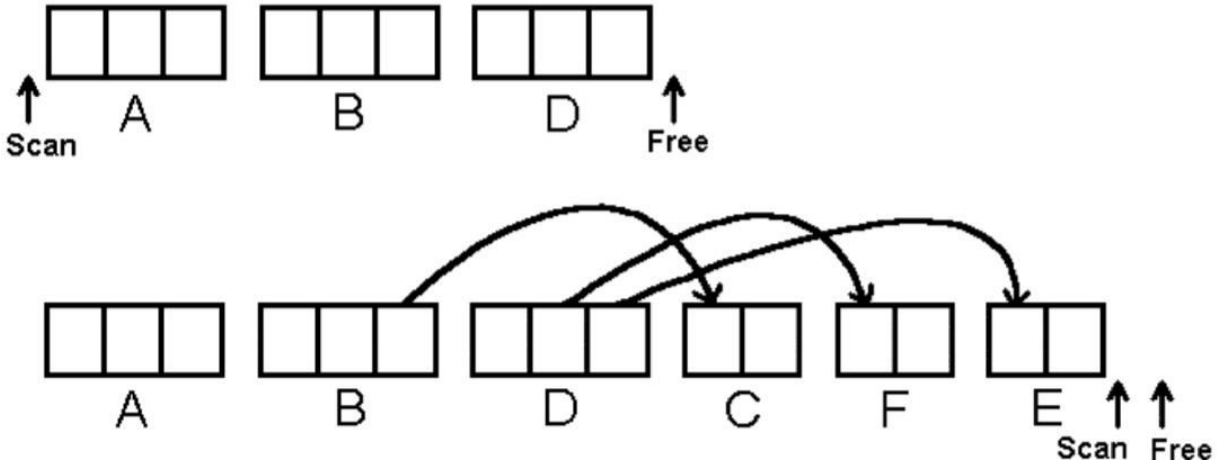
**• Form an initial queue of objects which can be immediately reached from the root set.**
**• A "scan" pointer is advanced through the objects location by location. Every time a pointer into fromspace is encountered, the object the pointer refers to is copied to the end of the queue.**
**•When the "scan" reaches the end of the queue, all live objects have been copied, so the garbage collector is terminated.**

**PROS :**

**Cheney's algorithm has a key advantage over non-copying techniques for allocation timing because it does not require the use of a complete allocator, with freeing and coalescing**

**The allocation of free objects is simple and fast.**
**This method does not cause memory fragmentation, even when objects of different sizes are copied.**

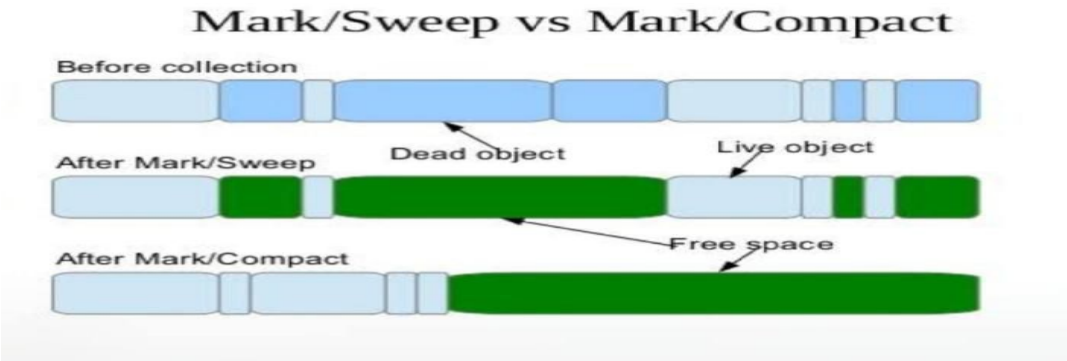
**OPTIMIZATION:To increase copying collectors efficiency, increase the amount of memory allocated for the heap space to reduce the number of times the collector is invoked.**




**(2) <u>MARK COMPACT ALGORITHM ?</u>**

**This algorithm is essentially a variaton on the Mark-Sweep algorithm just described.**
**Algorithm**

**In computer science, a mark-compact algorithm is a type of garbage collection algorithm used to reclaim unreachable memory. Mark-compact algorithms can be regarded as a combination of the mark-sweep algorithm and Cheney's copying algorithm. First, reachable objects are marked, then a compacting step relocates the reachable (marked) objects towards the beginning of the heap area. Compacting garbage collection is used by Microsoft's Common Language Runtime and by the Glasgow Haskell Compiler.**

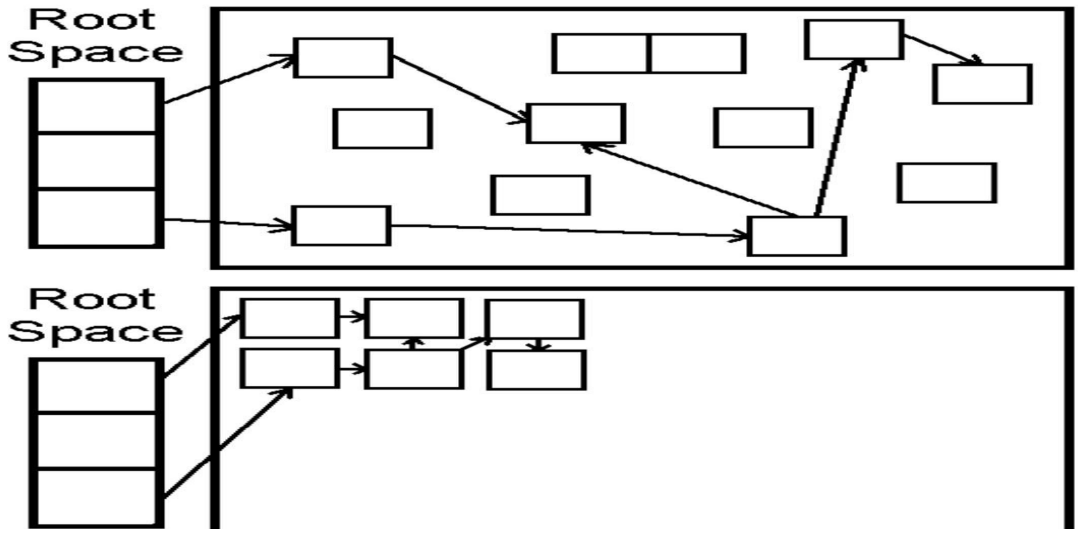


**PROS**

**The fragmentation problem of Mark-Sweep collection is solved with this algorithm; avaliable memory is put in a big single chunk.**
**Also note that the relative ordering of objects in memory stays the same - that is, if object X has a higher memory address than Y before garbage collection, it will still have a higher address afterwards. This property is important for certain data structures like arrays.**

**CONS**

**The big problem with Mark-Compact collection is time. It requires even more time than Mark-Sweep collection, which can seriously affect performance.**




**COMPARISON OF CHENEY'S AND MARK COMPACT ALGORITHM :**

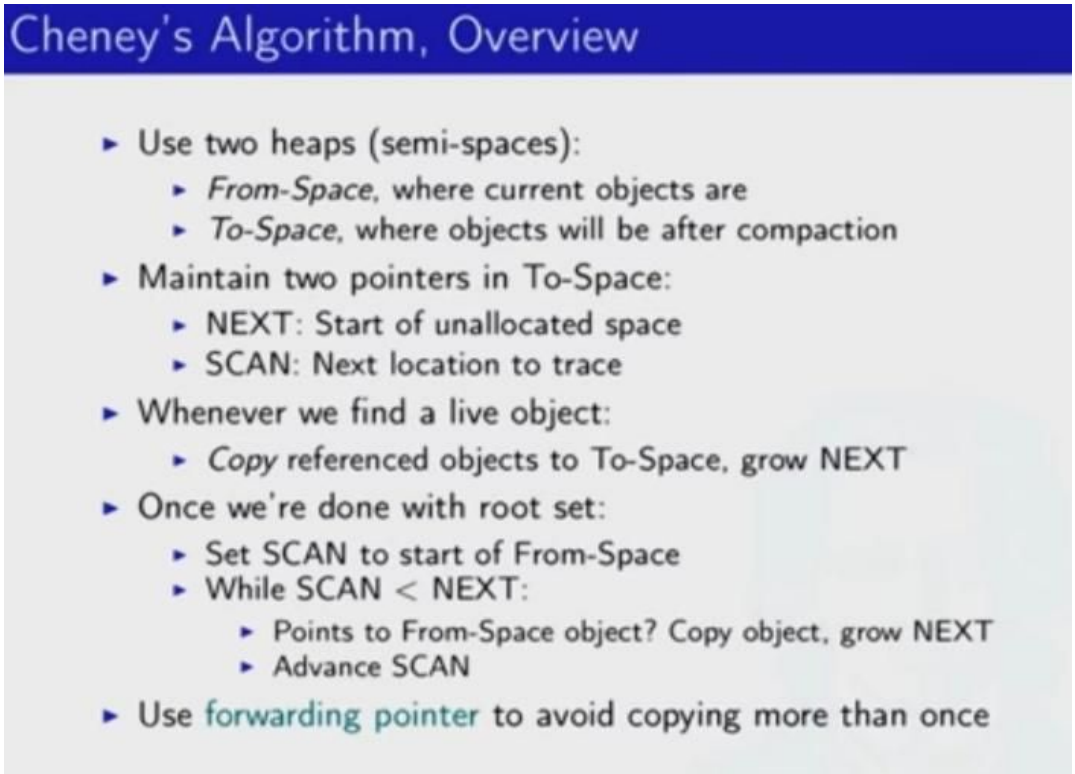
| CHENEY'S ALGORITHM | MARK COMPACT ALGORITHM |
|---|---|
| Fragmentation is minimized | Fragmentation is eliminated |
| Allocation is extremely cheap and easier | Allocation is fast and sequential |
| live data fromcurrent space is copied tothe other space, and then theroles are reversed | live objects are compacted together, the rest of the space is free |

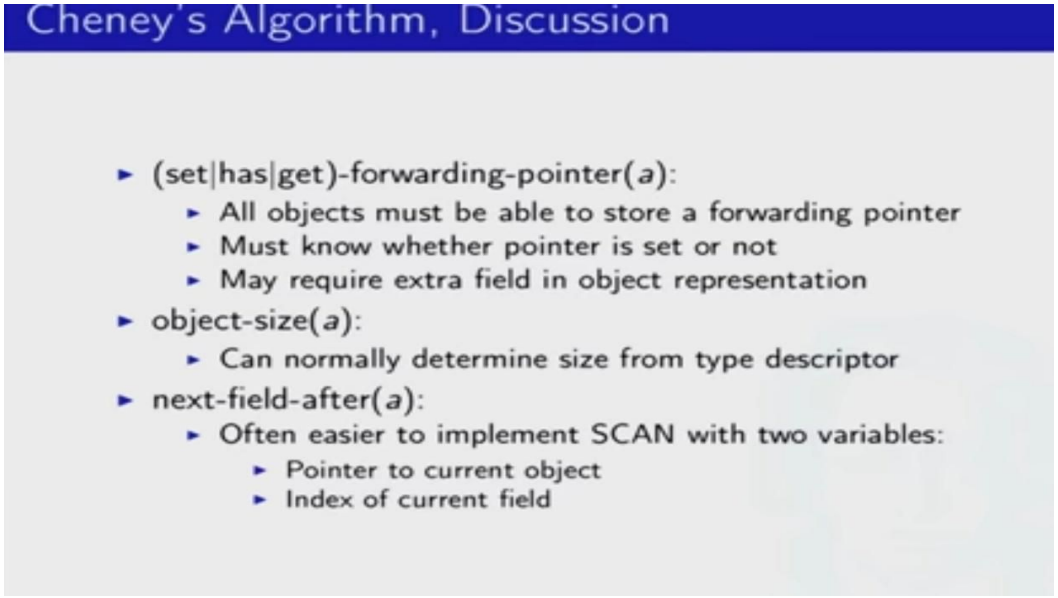
| Desired locality | Improved locality |
|---|---|

**COMPARISON OF CHENEY'S ALGORITHM AND MARK COMPACT ALGORITHM WITH C++**

**• C++ doesn't need a garbage collector, because it has no garbage. In modern C++ you use smart pointers and therefore have no garbage.**
**• The process of removing unused objects from heap memory is known as Garbage collection and this is a part of memory management in Java. Languages like C/C++ don't support automatic garbage collection, however in java, the garbage collection is automatic.**
**But, garbage collections can be used in c++ at some point.**

**CHENEY'S ALGORITHM WITH C++**






```
#include <iostream.h>
using namespace std;
Int main()
initialize() =
   tospace   = M/2
   fromspace = 0
   allocPtr  = fromspace
allocate(n) =
   If allocPtr + n > fromspace+ M/2
      collect()
   EndIf
   If allocPtr + n > fromspace+ M/2
      fail “insufficient memory”
   EndIf
   o = allocPtr
   allocPtr = allocPtr + m
   return o
collect() =
   swap(fromspace, tospace)
   allocPtr = fromspace
   scanPtr  = fromspace
elsewhere
      root = copy(root)
   EndForEach
```

```
   -- scan objects in the heap
   While scanPtr < allocPtr
         r = copy(r)
      EndForEach
      scanPtr = scanPtr  + o.size()
points to the next object in the heap, if any
   EndWhile

copy(o) =
   If o has no forwarding address
      o' = allocPtr
      allocPtr = allocPtr + size(o)
      copy the contents of o to o'
      forwarding-address(o) = o'
   EndIf
   return forwarding-address(o)
```

***MARK COMPACT ALGORITHM WITH C++***

```
#include<iostream.h>
using namespace std;
Int main()
struct Object {
   Object* head;
   Object* tail;
   bool marked;
};

const int HEAP_SIZE = 10000;
Object heap[HEAP_SIZE];
Object* root = nullptr;   // compile with -std=c++11 to get 'nullptr'
Object* free_list = nullptr;

void add_to_free_list(Object* p) {
   p->tail = free_list;
   free_list = p;
}
```

```
void init_heap() {
   for (int i = 0; i < HEAP_SIZE; i++)
      add_to_free_list(&heap[i]);
}

void mark(Object* p) {  // set the mark bit on p and all its descendants
   if (p == nullptr || p->marked)
      return;
   p->marked = true;
   mark(p->head);
   mark(p->tail);
}

Object* allocate() {
   if (free_list == nullptr) {      // out of memory, do GC
      for (int i = 0; i < HEAP_SIZE; i++)  // 1. clear mark bits
         heap[i].marked = false;
      mark(root);                        // 2. mark phase
      free_list = nullptr;                 // 3. sweep phase
      for (int i = 0; i < HEAP_SIZE; i++)
         if (!heap[i].marked)
            add_to_free_list(&heap[i]);
      if (free_list == nullptr)
         return nullptr;          // still out of memory :(
   }
   Object* p = free_list;
   free_list = free_list->tail;
   p->head = nullptr;
   p->tail = nullptr;
   return p;
}
```

**CONCLUSION :**

**I conclude that mark compact algorithm is better than Cheney's algorithm eventhough Cheney's algorithm is more effective. Mark compact is advanced and combination of mark sweep algorithm and Cheney's copying algorithm.whereas Cheney's algorithm is twice the heap size which is the major drawback and need**

**forwarding pointer. So mark compact algorithm  is far better than Cheney's algorithm from my opinion.**

**REFERENCES :**


- **Byers, Rick (2007). "Garbage Collection Algorithms" (PDF). Garbage Collection Algorithms. 12 (11): 3–4.**
- **Cheney, C.J. (November 1970). "A Nonrecursive List Compacting Algorithm". Communications of the ACM. 13 (11): 677–678. doi:10.1145/362790.362798**
- **https://en.m.wikipedia.org/wiki/Mark-compact_algorithm**


**PAPER REFERENCE :**


- **https://studylib.net/doc/14986660/gc-notes-by-bogdan-stroe**


**PERSONAL INFORMATION**


**Name : RENA NIRUSHIYA DAVID**
**Mail id: renanirushiyadavid18bca046@skasc.ac.in**