ANSWER TO THE QUESTION NUMBER 1

BUBBLE SORT:

এর টাইম কমপ্লেক্সযেটি হলো O(n^2)(worst case) কারন আমরা জানি বাবল শর্ট এর ক্ষেত্রে প্রথমে n টাইম একটি loop চলে এবং এর ভিতর আরও একটি loop চলে।

INSERTION SORT:

এক্ষেত্রে প্রথমে একটি n টাইম রুপ চলে ।এবং ভিতরে আরেকটি loop চলে। এর টাইম কমপ্লেক্সযেটি হলো

Best case >O(n)

কারণ সর্বশেষ উপাদানটিকে যদি স্থান পরিবর্তন করার দরকার না পরে।

Worst case >O(n^2)

এমন যদি হয় সর্বশেষ উপাদানটিকে একদম প্রথমে বসাতে হবে।

MERGE SORT:

এর time complexity হল O(nlogn)।

কারণ এখানে প্রতিবারই এরেকে দুই ভাগে ভাগ করা হচ্ছে। এর ফলে মোট লেভেল সংখ্যা হচ্ছে nএর বর্গমূল আবার প্রতি লেভেলেই কাজ হচ্ছে n time তাই time complexity হলো nlogn।

ANSWER TO THE QUESTION NUMBER 2

Array এবং **linked list** মধ্যে পার্থক্য **>**

i) আমরা যে সাইজের array ঘোষনা করি compiler মেমোরি থেকে ঠিক সেই পরিমাণ স্পেস ব্লক আকারে মেমোরি থেকে গ্রহণ করে থাকে অর্থাৎ অ্যারে এর indexগুলো মেমোরিতে একটি পর একটি থাকে।
যেমন, int array এর বেলায়
প্রথম index 20 এ হলে দ্বিতীয় টি হবে ২৪ এ তৃতীয় টি হবে ২৮ এভাবে চলবে

অপরদিকে লিংকড লিস্টে এরকম কোন বাধ্যবাধকতা নেই এগুলো মেমোরির যে কোন জায়গায় থাকতে পারে এবং পয়েন্টার দ্বারা একটি সাথে আরেকটি কানেক্ট থাকে।

ii) এরে থেকে আমরা যদি নির্দিষ্ট কোন index এর এলিমেন্টকে খুঁজে বের করতে চাই তাহলে time complexity হলো O(1)

কিন্তু linked list বেলায় তা O(n)

যে কারণে আমাদের **linked list** এ **head** বা **root** পয়েন্টার দরকার **>**

আমরা জানি linked list এ প্রতিটি নোড তারপর পরবর্তী নোড এর সাথে next নামক নোড পয়েন্টার দ্বারা কানেক্ট থাকে। এর মাধ্যমে আমরা ১ টির পর একটি নোডকে এক্সেস করি।
কিন্তু এই প্রক্রিয়াটা শুরু করার জন্য আমাদেরকে সর্বপ্রথম index 0 নোড কে এক্সেস করতে হবে।

আর ইনডেক্স জিরো এর নোড কে আমরা head/root পয়েন্টার ধারা পয়েন্ট করে রাখি। যাতে করে খুব সহজে আমরা index জিরোকে অ্যাক্সেস করতে পারি।

ANSWER TO THE QUESTION NUMBER 3

Basic idea of binary search algorithm >
প্রথমে এর এটিকে sort করতেহবে।
উদাহরণ স্বরূপ ধরে নেই আমাদের ছোট থেকে বড় ক্রমানুসারে একটি অ্যারে রয়েছে। এবং এ থেকে আমাদের 19 index খুঁজে বের করতে হবে।
[5,8,9,10,14,17,17,19,20]

এক্ষেত্রে আমরা প্রথমে এরে এর মিডেল ইনডেক্স কে এক্সেস করব। middle index যেহেতু 14 তাই বোঝা যাচ্ছে 14 এর আগের এলিমেন্ট গুলো গ্রহণযোগ্য নয়। এখন আমরা 14 পরের আংশ নিয়ে কাজ করবো এবং পূর্ববর্তী অর্ধেক অংশকে ফেলে দিব। তারপরের অংশের মিডেল ইনডেক্স এর element হলো 17।তাই আমরা এ অংশ ফেলে দিয়ে 17 এর পরের অংশ নিয়ে কাজ করব। আমরা দেখতে পাচ্ছি বাইনারি search এ প্রতি অপারেশনে অর্ধেক অ্যারে ফেলে দেওয়া হয়। এবং কাঙ্খিত উপাদানটি নেওয়া পাওয়া পর্যন্ত লাস্ট ইনডেক্স পর্যন্ত অপারেশন চলে।

Difference between binary and linear search >
i) linear search sorted কিংবা unsorted উভয় ধরনের array তে। কাজ চালাতে পারে উভয় ধরনের কিন্তু binary search শুধুমাত্র
sorted array তে কাজ চালাতে পারে।

ii) linear search এর time complexity O(n)।কিন্তু binary search এর time complexity O(logn)।

কোনো array তে বাইনারি সার্চ করার পূর্ব শর্ত >
Array টিকে অবশ্যই sorted অবস্থায় থাকতে হবে।

ANSWER TO THE QUESTION NUMBER 4

i) linked list এর শুরুতে কোন উপাদান add করতে চাইলে time complexity হবেO(1)।

ii) linked list এর index এ কোন উপাদান add করতে চাইলে time complexity হবেO(n)।

iii) linked list এর শুরুতে কোন উপাদান delete করতে চাইলে time complexity হবেO(1)।

iv) linked list এর যেকোনো index এর কোন উপাদান delete করতে চাইলে time complexity হবেO(n)।

ANSWER TO THE QUESTION NUMBER 5

singly linked list দ্বিগুণ মেমোরি স্পেস নেবে এর কারণ হলো singly linked list এর প্রতিটি নোডে দুটি করে ভেরিয়েবল থাকবে একটি হলো তার ডাটা অপরটি হল একটি পয়েন্টার টাইপ ভেরিয়েবল যেটি তার পরবর্তী নোড কে পয়েন্ট করে রাখবে।

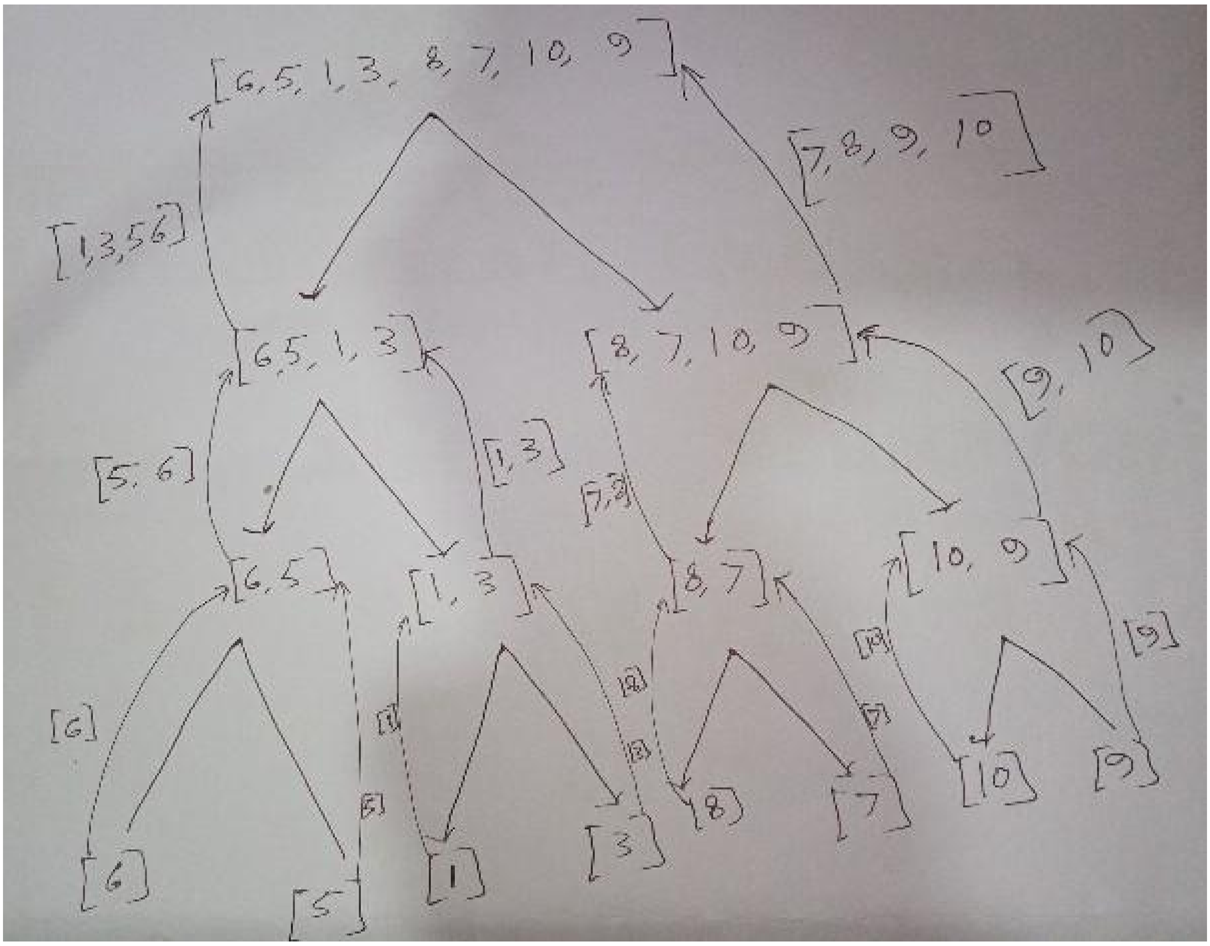
অপরদিকে array তে তার প্রতিটি ইনডেক্সে একটি করে ডাটা থাকবে এর কারণে singly linked list array থেকে দ্বিগুণ মেমোরি নিবে।

ANSWER TO THE QUESTION NUMBER 6

Worst case time complexity O(n^2)।
কারন এটি নির্ভর করে pivot choice করার ওপর।
ধরা যাক [1,2,4,5,6,7,] অ্যারেতে last index কে pivot হিসাবে ধরলাম তাহলে মোট লেভেল সংখ্যা হবে n
আবার প্রতি লেভেলে যেহেতু n loop চলবে তাই time complexity O(n^2)।

Average case time complexity O(nlogn)।
ধরা যাক [1,2,4,5,6,7,] অ্যারেতে middle index কে pivot হিসাবে ধরলাম তাহলে মোট লেভেল সংখ্যা হবে logn ।এক্ষেত্রে প্রতি lavel এ অ্যার অর্ধেক হয়ে যাচ্ছে এবং logn level পর আমরা 1 বা 0 size এর array পাবো।
আবার প্রতি লেভেলে যেহেতু n loop চলবে তাই time complexity O(nlogn)।

ANSWER TO THE QUESTION NUMBER 7



ANSWER TO THE QUESTION NUMBER 8

উক্ত program টিতে আমরা দেখতে পাচ্ছি n এর মান যত হবে loop টি ১থেকে তার বর্গমূল বার চলবে।

অর্থাৎ n মান যদি 25 loop চলবে 5 বার আবার n এর মান 20 হলে loop চলবে 4(শুধুমাত্র পূর্ণ সংখ্যাটুকু নেয়া হবে) বার।

তাই বলতে পারি এর টাইম কমপ্লেক্সিটি logn

ANSWER TO THE QUESTION NUMBER 9

এ ক্ষেত্রে array ভালো হবে কারন আমরা জানি array এর প্রতিটি উপাদান একটির পর একটির পাশাপাশি অবস্থান করে যার কারনে কোনো random index কে access করতে হলো শুদু মাত্র তার index নাম্বার টি হলেই O(1) time complexity তেই একে access করা যায়া কিন্তু linked list এ এর জন্য time comexity হবে O(n)। আবার array তে বাইনারি search বাস্তবায়ন করা খুব সহজ linked list এর তুলনায়,তাই linked list এ সাধারনত linear search করা করা হয়।

এসব দিক বিবেচনা করে আমরা বলতে পারি উক্ত project এ array ব্যাবহার করা অধিক সুবিধাজনক হবে linked list এর তুলনায়।

ANSWER TO THE QUESTION NUMBER 10

—>

কোনো text editor এ আমরা যখন কোন টেক্স ডিলেট করে দেয়ার পর আমাদের যদি মনে হয় না এ টেক্সটি আমাদের দরকার তখন আমরা undo operation এর মাধ্যমে ডিলিট করা সেই টেক্সটকে ফিরিয়ে আনতে পারি।

কিন্তু এরপরে আমাদের যদি মনে হয় টেক্সটি ডিলিট করাই দরকার অর্থাৎ আমরা একে undo করার পূর্বের অবস্থায় ফিরিয়ে নিতে চাই এর জন্য
আমরা redo করবো।

অর্থাৎ redo operation মাধ্যমে আমরা undo এর পূর্বের অবস্থায় ফিরে যেতে পারি।
অর্থাৎ redo operation করতে হলে আমাদের এর পূর্বে একটি undo operation থাকতে হবে।
তাহলে আমরা দেখতে পাচ্ছি undo-redo operation ভুল উদ্দেশ্য হলো পূর্বের অবস্থায় ফিরে যাওয়া কিন্তু singly linked list এর ক্ষেত্রে আমরা জানি next pointer ধারা আমরা শুধুমাত্র সামনের দিকে অগ্রসর হতে পারব। অপরদিকে doubly linked list এ আমরা জানি pre এবং next দুটি পয়েন্টার থাকে যার কারণে সামনে এবং পিছনে দুটি ডিরেকশনে চলা খুবই সহজ।
অর্থাৎ doubly linked list এর মাধ্যমে খুব সহজেই পূর্বের অবস্থার অ্যাক্সেস নেওয়া যায়।
তাই আমরা বলতে পারি উক্ত প্রজেক্ট এর জন্য singly linked list অপেক্ষা doubly linked list অধিক সুবিধা জনক