```
thisis = "2016-06-20 Quantum computing.nb";
(* Sample Mathematica code (all but Shor's algorithm) for article
 "Undergraduate computational physics projects on quantum computing"
 by D. Candela.

  Copyright (C)2016 Donald Candela

 This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version. This
 program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

  Random notes:
  1. This file contains code for all of the projects except Shor's algorithm,
which is in the separate file "2016-XX-XX Shor algorithm.nb".
  2. This code was written to check
 the formulas and provide output for the article;
it is not intended as sample code to be provided to students.  The
 point of the article is that students should write their own code.
  3. Mathematica is more obscure and fussy than most other computer languages,
and is not the preferred platform for carrying out these projects.  Unless
 you are already extremely comfortable with Mathematica (as I was)
 it would be better to use another platform like Python or C++.
  4. Array indices in Mathematica start with 1
 (like MATLAB, unlike Python or C++).  Therefore in this code
 one is subtracted from array indices before computations,
and added back to computation results before being used as array indices.
  5. The Mathematica functions IntegerDigits[] and FromDigits[] are used here to
 convert integers into lists of their binary digits and vice versa. These
 functions may need to be written from scratch in other computer languages.
  6. All variables are given lowercase names.  Letters are
 doubled to indicate variables that are uppercase in the
 article.  Thus the number of qubits N is called nn in this code.
  7. All of this code is written for an arbitary number of qubits N,
unlike the earlier sections of the article which are written for N=3 only.
   8. References to sections and formulas
  in the article are in square brackets eg [Eq 34].
   9. Mathematica's built in sparse matrix functions are not used here;
```

```
instead sparse matrix functions are written out to provide
 an example of how to do this.  The built in sparse matrix
 functions might be much more efficient than the ones used here.
  10. Mathematica becomes very slow when any array dimension
 becomes large (10^5 - 10^6).  Therefore as the number of
 qubits is increased towards the maximum these algorithms may
 execute much faster in a more conventional language like C++.
 *)



(* Before any of the following functions are
  called the global nn must be set to the number of qubits.
   The basis vectors are numbered from j=
 1 meaning |0000> to j=2^nn meaning |11111>. *)

(* Utility function returns string "|bbb bbbb>" denoting jth basis vector. *)
bstring[j_] := Module[{digits, d},
   digits = Map[ToString, IntegerDigits[j - 1, 2, nn]];

   (* add a space every 4th digit *)
   For[d = 1, d ≤ Length[digits], d++,
    If[Divisible[d, 5], digits = Insert[digits, " ", -d]]];

   "|" <> StringJoin[digits] <> "⟩"
  ];

(* Function performs mm measurements on
 the state vector psi and prints the results *)
measure[psi_, mm_] := Module[{dd, hist, perline, m, mstring, j, tstring},
   dd = 2^nn;
   hist = Table[{0, j}, {j, dd}]; (* collect histogram of results *)
   perline = Floor[80 / (StringLength[bstring[1]] + 1)];
   (* how many measurements to print on one line *)
   mstring = "Measurements:\n";
   For[m = 1, m ≤ mm, m++,
    j = measure1[psi];
    hist[[j, 1]]++;
    mstring = mstring <> " " <> bstring[j];
    If[Divisible[m, perline], mstring = mstring <> "\n"];
   ];
   Print[mstring];
   hist = SortBy[hist, First];
```

```
   tstring = "Most frequent results: ";
   For[j = 1, j ≤ 4, j++, tstring =
     tstring <> bstring[hist[[-j, 2]]] <> "(" <> ToString[hist[[-j, 1]]] <> ") "];
   Print[tstring]
  ];

(* Function performs a single measurement, returns j when the
 result is the jth basis vector [Appendix, Hints for project 1] *)
measure1[psi_] := Module[{dd, r, q, j},
   dd = 2^nn;
   r = RandomReal[]; (* random number in (0,1) *)
   j = 1;
   q = Conjugate[psi[[1]]] * psi[[1]]; (* cumulative probability *)
   While[r > q && j <= dd,
    j++; q += Conjugate[psi[[j]]] * psi[[j]]];
   j];

(* PROJECT 1a: Initialize to a basis state, apply no gates *)
nn = 3;(* number of qubits *)
j = 3;(* basis state to initilize to *)
mm = 100;(* number of measurements to do *)

Print["N = ", nn, ", Inital state = ", bstring[j]];
Print["Comptutation: None"];
dd = 2^nn;
psi = Table[0, {dd}];
psi[[j]] = 1;
measure[psi, mm];
```

```
N = 3, Inital state = |010⟩

Comptutation: None

Measurements:
 |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩
 |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩
 |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩
 |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩
 |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩
 |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩
 |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩
 |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩

Most frequent results: |010⟩(100) |111⟩(0) |110⟩(0) |101⟩(0)
```

```
(* PROJECT 1b: Initialize to cat state, apply no gates *)
nn = 5;(* number of qubits *)
mm = 100;(* number of measurements to do *)

Print["N = ", nn, ", Inital state = cat"];
Print["Comptutation: None"];
dd = 2^nn;
psi = Table[0, {dd}];
s2 = 1 / Sqrt[2.];
psi[[1]] = s2;
psi[[dd]] = s2;
measure[psi, mm];
```

```
N = 5, Inital state = cat

Comptutation: None

Measurements:
 |1 1111⟩ |1 1111⟩ |1 1111⟩ |1 1111⟩ |1 1111⟩ |0 0000⟩ |1 1111⟩ |0 0000⟩
 |0 0000⟩ |0 0000⟩ |0 0000⟩ |1 1111⟩ |0 0000⟩ |0 0000⟩ |1 1111⟩ |1 1111⟩
 |1 1111⟩ |1 1111⟩ |1 1111⟩ |0 0000⟩ |1 1111⟩ |1 1111⟩ |0 0000⟩ |0 0000⟩
 |1 1111⟩ |0 0000⟩ |0 0000⟩ |0 0000⟩ |1 1111⟩ |1 1111⟩ |1 1111⟩ |1 1111⟩
 |1 1111⟩ |1 1111⟩ |0 0000⟩ |0 0000⟩ |0 0000⟩ |1 1111⟩ |1 1111⟩ |1 1111⟩
 |1 1111⟩ |1 1111⟩ |1 1111⟩ |1 1111⟩ |0 0000⟩ |1 1111⟩ |0 0000⟩ |1 1111⟩
 |0 0000⟩ |1 1111⟩ |1 1111⟩ |0 0000⟩ |0 0000⟩ |0 0000⟩ |1 1111⟩ |1 1111⟩
 |0 0000⟩ |1 1111⟩ |0 0000⟩ |0 0000⟩ |1 1111⟩ |1 1111⟩ |0 0000⟩ |0 0000⟩
 |1 1111⟩ |0 0000⟩ |1 1111⟩ |1 1111⟩ |1 1111⟩ |1 1111⟩ |1 1111⟩ |1 1111⟩
 |0 0000⟩ |1 1111⟩ |1 1111⟩ |1 1111⟩ |1 1111⟩ |0 0000⟩ |1 1111⟩ |0 0000⟩
 |0 0000⟩ |1 1111⟩ |0 0000⟩ |1 1111⟩ |0 0000⟩ |1 1111⟩ |0 0000⟩ |0 0000⟩
 |0 0000⟩ |1 1111⟩ |1 1111⟩ |0 0000⟩ |0 0000⟩ |1 1111⟩ |0 0000⟩ |0 0000⟩
 |0 0000⟩ |1 1111⟩ |1 1111⟩ |0 0000⟩

Most frequent results: |1 1111⟩(57) |0 0000⟩(43) |1 1110⟩(0) |1 1101⟩(0)
```

```
(* PROJECT 1c: Initialize to equal superposition of all basis states,
apply no gates *)
nn = 3;(* number of qubits *)
mm = 300;(* number of measurements to do *)

Print["N = ", nn, ", Inital state = equal superposition of all basis states"];
Print["Comptutation: None"];
dd = 2^nn;
sdd = N[1 / Sqrt[dd]];
psi = Table[sdd, {dd}];
measure[psi, mm];
```

N = 3, Inital state = equal superposition of all basis states

Comptutation: None

Measurements:

|011⟩ |000⟩ |101⟩ |000⟩ |100⟩ |110⟩ |110⟩ |110⟩ |000⟩ |101⟩ |100⟩ |001⟩ |000⟩
|001⟩ |010⟩ |110⟩ |100⟩ |100⟩ |110⟩ |000⟩ |110⟩ |000⟩ |001⟩ |110⟩ |011⟩ |101⟩
|001⟩ |110⟩ |011⟩ |111⟩ |110⟩ |001⟩ |010⟩ |101⟩ |111⟩ |010⟩ |111⟩ |100⟩ |010⟩
|000⟩ |101⟩ |010⟩ |011⟩ |110⟩ |101⟩ |011⟩ |101⟩ |001⟩ |000⟩ |101⟩ |011⟩ |010⟩
|100⟩ |111⟩ |011⟩ |101⟩ |000⟩ |000⟩ |101⟩ |101⟩ |110⟩ |001⟩ |111⟩ |100⟩ |000⟩
|001⟩ |100⟩ |010⟩ |001⟩ |000⟩ |100⟩ |110⟩ |010⟩ |001⟩ |111⟩ |010⟩ |000⟩ |110⟩
|111⟩ |000⟩ |101⟩ |010⟩ |100⟩ |000⟩ |000⟩ |101⟩ |101⟩ |011⟩ |001⟩ |110⟩ |101⟩
|001⟩ |001⟩ |110⟩ |110⟩ |110⟩ |011⟩ |000⟩ |101⟩ |100⟩ |100⟩ |011⟩ |111⟩ |001⟩
|001⟩ |011⟩ |100⟩ |000⟩ |011⟩ |100⟩ |110⟩ |000⟩ |011⟩ |010⟩ |000⟩ |101⟩ |110⟩
|010⟩ |010⟩ |000⟩ |001⟩ |000⟩ |111⟩ |000⟩ |010⟩ |011⟩ |011⟩ |111⟩ |011⟩ |111⟩
|011⟩ |001⟩ |001⟩ |011⟩ |001⟩ |111⟩ |011⟩ |000⟩ |101⟩ |110⟩ |000⟩ |111⟩ |101⟩
|101⟩ |010⟩ |101⟩ |111⟩ |101⟩ |010⟩ |100⟩ |011⟩ |010⟩ |011⟩ |001⟩ |000⟩ |011⟩
|001⟩ |110⟩ |110⟩ |100⟩ |111⟩ |101⟩ |110⟩ |100⟩ |000⟩ |101⟩ |110⟩ |111⟩ |111⟩
|101⟩ |100⟩ |100⟩ |000⟩ |110⟩ |101⟩ |111⟩ |011⟩ |100⟩ |101⟩ |010⟩ |100⟩ |010⟩
|110⟩ |110⟩ |001⟩ |011⟩ |000⟩ |011⟩ |100⟩ |010⟩ |010⟩ |010⟩ |010⟩ |000⟩ |001⟩
|100⟩ |011⟩ |100⟩ |110⟩ |010⟩ |100⟩ |111⟩ |000⟩ |001⟩ |000⟩ |001⟩ |010⟩ |111⟩
|010⟩ |011⟩ |110⟩ |010⟩ |111⟩ |100⟩ |000⟩ |011⟩ |100⟩ |110⟩ |010⟩ |101⟩ |010⟩
|100⟩ |101⟩ |110⟩ |011⟩ |110⟩ |001⟩ |100⟩ |000⟩ |011⟩ |110⟩ |010⟩ |111⟩ |110⟩
|001⟩ |010⟩ |000⟩ |000⟩ |110⟩ |101⟩ |000⟩ |100⟩ |101⟩ |001⟩ |010⟩ |110⟩ |011⟩
|111⟩ |010⟩ |100⟩ |100⟩ |110⟩ |010⟩ |001⟩ |111⟩ |010⟩ |101⟩ |010⟩ |110⟩ |101⟩
|111⟩ |101⟩ |111⟩ |111⟩ |101⟩ |100⟩ |001⟩ |000⟩ |100⟩ |011⟩ |000⟩ |100⟩ |101⟩
|100⟩ |011⟩ |100⟩ |011⟩ |101⟩ |010⟩ |101⟩ |111⟩ |110⟩ |011⟩ |110⟩ |100⟩ |011⟩
|100⟩ |011⟩ |010⟩ |110⟩ |101⟩ |100⟩ |100⟩ |111⟩ |111⟩ |100⟩ |101⟩ |111⟩ |101⟩
|101⟩

Most frequent results: |101⟩(42) |100⟩(42) |110⟩(41) |000⟩(39)

```
(* FOR ROJECT 2: Implement Hadamard and Phase-Shift Gates. *)

(* One-qubit Hadamard matrix *)
hadamard = (1 / Sqrt[2.]) * {{1, 1}, {1, -1}};

(* Function returns one-qubit phase-shift gate *)
phaseshift[theta_] := Module[{},
   {{1, 0}, {0, N[Exp[I * theta]]}}];

(* Function takes matrix for a one-bit gate g1 and qubit number q;
returns matrix g1 applied to qubit q in an nn-
 qubit system [Eq 34 and Hints for project 5] *)
```

```
expand1[g1_, q_] := Module[{dd, matq, j, jbits, jq, k, kbits, kq},
   dd = 2^nn;
   matq = Table[0, {dd}, {dd}]; (* start with matrix all zeroes *)
   For[j = 1, j ≤ dd, j++,
    jbits = IntegerDigits[j - 1, 2, nn]; (* binary digits for j *)
    jq = jbits[[q]];
    (* bit of j that will index the one-qubit matrix *)
    For[k = 1, k ≤ dd, k++,
     kbits = IntegerDigits[k - 1, 2, nn]; (* binary digits for k *)
     kq = kbits[[q]];
     (* bit of k that will index the one-qubit matrix *)
     kbits[[q]] = jbits[[q]];
     (* this prevents kronecker enforement for bit q *)
     If[jbits == kbits, (* this enforces the kronecker deltas *)
      matq[[j, k]] = g1[[jq + 1, kq + 1]];
     ]
    ];
   ];
   matq
  ];

(* Generate 3-
 qubit matrices and print out to check against article [Eqs(20-23)] *)
nn = 3;
had1 = expand1[hadamard, 1];
had2 = expand1[hadamard, 2];
had3 = expand1[hadamard, 3];
pi3 = expand1[phaseshift[Pi], 3];
Print["H1 = ", had1 // MatrixForm];
Print["H2 = ", had2 // MatrixForm];
Print["H3 = ", had3 // MatrixForm];
Print["Pi3 = ", pi3 // MatrixForm];
```

H1 =

$$\begin{pmatrix} 0.707107 & 0 & 0 & 0 & 0.707107 & 0 & 0 & 0 \\ 0 & 0.707107 & 0 & 0 & 0 & 0.707107 & 0 & 0 \\ 0 & 0 & 0.707107 & 0 & 0 & 0 & 0.707107 & 0 \\ 0 & 0 & 0 & 0.707107 & 0 & 0 & 0 & 0.707107 \\ 0.707107 & 0 & 0 & 0 & -0.707107 & 0 & 0 & 0 \\ 0 & 0.707107 & 0 & 0 & 0 & -0.707107 & 0 & 0 \\ 0 & 0 & 0.707107 & 0 & 0 & 0 & -0.707107 & 0 \\ 0 & 0 & 0 & 0.707107 & 0 & 0 & 0 & -0.707107 \end{pmatrix}$$

H2 =

$$\begin{pmatrix} 0.707107 & 0 & 0.707107 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.707107 & 0 & 0.707107 & 0 & 0 & 0 & 0 \\ 0.707107 & 0 & -0.707107 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.707107 & 0 & -0.707107 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.707107 & 0 & 0.707107 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.707107 & 0 & 0.707107 \\ 0 & 0 & 0 & 0 & 0.707107 & 0 & -0.707107 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.707107 & 0 & -0.707107 \end{pmatrix}$$

$$H3 = \begin{pmatrix} 0.707107 & 0.707107 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.707107 & -0.707107 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.707107 & 0.707107 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.707107 & -0.707107 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.707107 & 0.707107 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.707107 & -0.707107 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.707107 & 0.707107 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.707107 & -0.707107 \end{pmatrix}$$

$$Pi3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1. & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1. & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1. & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1. \end{pmatrix}$$

```
(* PROJECT 2a [Fig 3(a)]: Initialize to |000>; Hadamard on qubit 2 *)
mm = 100;(* number of measurements to do *)

Print["N = ", nn, ", Inital state = ", bstring[1]];
Print["Comptutation: Hadamard on qubit 2"];
dd = 2^nn;
psi = Table[0, {dd}];
psi[[1]] = 1;
psi = had2.psi;
measure[psi, mm];
```

N = 3, Inital state = |000⟩

Comptutation: Hadamard on qubit 2

Measurements:

|000⟩ |010⟩ |010⟩ |000⟩ |000⟩ |000⟩ |010⟩ |000⟩ |000⟩ |010⟩ |010⟩ |010⟩ |010⟩
|000⟩ |010⟩ |000⟩ |010⟩ |010⟩ |010⟩ |010⟩ |000⟩ |000⟩ |010⟩ |000⟩ |010⟩ |010⟩
|010⟩ |000⟩ |010⟩ |010⟩ |010⟩ |000⟩ |010⟩ |000⟩ |010⟩ |000⟩ |000⟩ |000⟩ |010⟩
|010⟩ |010⟩ |000⟩ |010⟩ |000⟩ |010⟩ |010⟩ |000⟩ |010⟩ |010⟩ |000⟩ |000⟩ |010⟩
|000⟩ |010⟩ |000⟩ |000⟩ |010⟩ |000⟩ |010⟩ |010⟩ |010⟩ |010⟩ |010⟩ |000⟩ |000⟩
|010⟩ |000⟩ |010⟩ |010⟩ |000⟩ |010⟩ |000⟩ |010⟩ |010⟩ |010⟩ |010⟩ |000⟩ |000⟩
|000⟩ |000⟩ |010⟩ |010⟩ |010⟩ |000⟩ |000⟩ |010⟩ |000⟩ |010⟩ |010⟩ |000⟩ |000⟩
|000⟩ |000⟩ |010⟩ |000⟩ |000⟩ |010⟩ |000⟩ |010⟩ |000⟩

Most frequent results: |010⟩(54) |000⟩(46) |111⟩(0) |110⟩(0)

```
(* PROJECT 2b [Fig 3(b)]: Initialize to |000>; Hadamard on qubits 1, 2, 3 *)
mm = 300;(* number of measurements to do *)

Print["N = ", nn, ", Inital state = ", bstring[1]];
Print["Comptutation: Hadamard on qubits 1,2 3"];
dd = 2^nn;
psi = Table[0, {dd}];
psi[[1]] = 1;
psi = had1.psi;
psi = had2.psi;
psi = had3.psi;
measure[psi, mm];
```

```
N = 3, Inital state = |000⟩

Comptutation: Hadamard on qubits 1,2 3

Measurements:
 |110⟩ |111⟩ |010⟩ |001⟩ |111⟩ |111⟩ |011⟩ |101⟩ |101⟩ |000⟩ |111⟩ |101⟩ |010⟩
 |101⟩ |111⟩ |000⟩ |100⟩ |001⟩ |000⟩ |110⟩ |011⟩ |001⟩ |011⟩ |111⟩ |011⟩ |000⟩
 |110⟩ |001⟩ |000⟩ |100⟩ |101⟩ |011⟩ |110⟩ |110⟩ |001⟩ |100⟩ |100⟩ |000⟩ |101⟩
 |111⟩ |111⟩ |010⟩ |001⟩ |111⟩ |001⟩ |111⟩ |010⟩ |101⟩ |011⟩ |111⟩ |000⟩ |111⟩
 |110⟩ |110⟩ |001⟩ |010⟩ |101⟩ |100⟩ |000⟩ |110⟩ |101⟩ |000⟩ |100⟩ |011⟩ |000⟩
 |100⟩ |110⟩ |101⟩ |101⟩ |101⟩ |100⟩ |011⟩ |000⟩ |011⟩ |111⟩ |100⟩ |101⟩ |101⟩
 |111⟩ |001⟩ |010⟩ |011⟩ |100⟩ |110⟩ |101⟩ |101⟩ |000⟩ |100⟩ |000⟩ |000⟩ |111⟩
 |101⟩ |000⟩ |100⟩ |010⟩ |000⟩ |100⟩ |011⟩ |001⟩ |101⟩ |101⟩ |011⟩ |101⟩ |111⟩
 |100⟩ |100⟩ |101⟩ |011⟩ |101⟩ |000⟩ |000⟩ |100⟩ |000⟩ |101⟩ |110⟩ |100⟩ |100⟩
 |111⟩ |101⟩ |100⟩ |100⟩ |111⟩ |001⟩ |001⟩ |010⟩ |101⟩ |010⟩ |100⟩ |011⟩ |110⟩
 |110⟩ |000⟩ |111⟩ |011⟩ |110⟩ |011⟩ |101⟩ |010⟩ |011⟩ |000⟩ |110⟩ |001⟩ |100⟩
 |010⟩ |110⟩ |000⟩ |011⟩ |010⟩ |001⟩ |101⟩ |101⟩ |100⟩ |010⟩ |100⟩ |110⟩ |100⟩
 |110⟩ |001⟩ |010⟩ |110⟩ |100⟩ |001⟩ |011⟩ |110⟩ |101⟩ |100⟩ |100⟩ |011⟩ |000⟩
 |011⟩ |001⟩ |010⟩ |010⟩ |001⟩ |010⟩ |111⟩ |101⟩ |010⟩ |110⟩ |101⟩ |111⟩ |111⟩
 |001⟩ |011⟩ |001⟩ |110⟩ |110⟩ |010⟩ |000⟩ |111⟩ |100⟩ |110⟩ |110⟩ |100⟩ |010⟩
 |110⟩ |000⟩ |101⟩ |111⟩ |101⟩ |001⟩ |001⟩ |110⟩ |000⟩ |001⟩ |101⟩ |100⟩ |000⟩
 |100⟩ |101⟩ |000⟩ |111⟩ |111⟩ |011⟩ |011⟩ |101⟩ |101⟩ |100⟩ |110⟩ |000⟩ |001⟩
 |011⟩ |100⟩ |110⟩ |000⟩ |110⟩ |000⟩ |101⟩ |101⟩ |010⟩ |000⟩ |110⟩ |100⟩ |010⟩
 |101⟩ |001⟩ |110⟩ |110⟩ |010⟩ |011⟩ |011⟩ |100⟩ |000⟩ |001⟩ |110⟩ |001⟩ |111⟩
 |000⟩ |111⟩ |111⟩ |111⟩ |101⟩ |010⟩ |100⟩ |110⟩ |111⟩ |111⟩ |001⟩ |111⟩ |100⟩
 |001⟩ |111⟩ |000⟩ |110⟩ |110⟩ |101⟩ |110⟩ |001⟩ |110⟩ |111⟩ |010⟩ |101⟩ |101⟩
 |000⟩ |111⟩ |011⟩ |000⟩ |100⟩ |100⟩ |001⟩ |100⟩ |101⟩ |010⟩ |010⟩ |001⟩ |111⟩
 |001⟩ |000⟩ |010⟩ |000⟩ |111⟩ |101⟩ |111⟩ |001⟩ |101⟩ |001⟩ |000⟩ |011⟩ |111⟩
 |101⟩

Most frequent results: |101⟩(48) |100⟩(41) |111⟩(40) |000⟩(40)
```

```
(* PROJECT 2c [Fig 3(c)]: Initialize to |000>; two Hadamards on qbit 3 *)
mm = 100;(* number of measurements to do *)

Print["N = ", nn, ", Inital state = ", bstring[1]];
Print["Comptutation: 2 Hadamards on qubit 3"];
dd = 2^nn;
psi = Table[0, {dd}];
psi[[1]] = 1;
psi = had3.psi;
psi = had3.psi;
measure[psi, mm];
```

```
N = 3, Inital state = |000〉

Comptutation: 2 Hadamards on qubit 3

Measurements:
 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉
 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉
 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉
 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉
 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉
 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉
 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉
 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉 |000〉

Most frequent results: |000〉(100) |111〉(0) |110〉(0) |101〉(0)
```

```
(* PROJECT 2d [Fig 3(d)]: Initialize to |000>;
Hadamard, pi shift, Hadamard on qubit 3 *)
mm = 100;(* number of measurements to do *)

Print["N = ", nn, ", Inital state = ", bstring[1]];
Print["Comptutation: Hadamard, pi shift, Hadamard on qubit 3"];
dd = 2^nn;
psi = Table[0, {dd}];
psi[[1]] = 1;
psi = had3.psi;
psi = pi3.psi;
psi = had3.psi;
measure[psi, mm];
```

```
N = 3, Inital state = |000⟩
Comptutation: Hadamard, pi shift, Hadamard on qubit 3
Measurements:
 |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩
 |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩
 |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩
 |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩
 |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩
 |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩
 |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩
 |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩ |001⟩
Most frequent results: |001⟩(100) |111⟩(0) |110⟩(0) |101⟩(0)
```

```
(* PROJECT 3: Grover search - definitions, etc prior to actual calculations *)

(* Function computes oracle matrix for correct question = jc [Eq 26]*)
comporacle[jc_] := Module[{dd, mat, j},
   dd = 2^nn;
   mat = Table[0, {dd}, {dd}]; (* start with matrix all zeroes *)
   For[j = 1, j ≤ dd, j++,
    If[j == jc, mat[[j, j]] = -1, mat[[j, j]] = 1];
   ];
   mat
  ];

(* Function carries out Grover quantum calculation
 for correct answer jc and iterations its [Fig 4] *)
compgrover[jc_, its_, mm_] := Module[{oracle, dd, psi, q, it},
   Print["N = ", nn, ", Inital state = ", bstring[1]];
   Print["Comptutation: Grover search with correct question = ",
    bstring[jc], " and ", its, " iterations"];
   Print["Computing oracle matrix took ", Timing[
      oracle = comporacle[jc];
     ][[1]], " sec"];

   Print["Performing quantum computation took ", Timing[
      (* set initial state = |000> *)
      dd = 2^nn;
      psi = Table[0, {dd}];
      psi[[1]] = 1;
```

```
      (* perform intial Hadamard on each qubit *)
      For[q = 1, q ≤ nn, q++, psi = hadn[[q]].psi];

      (* do Grover steps *)
      For[it = 1, it ≤ its, it++,
       psi = oracle.psi;
       For[q = 1, q ≤ nn, q++, psi = hadn[[q]].psi];
       psi = jj.psi;
       For[q = 1, q ≤ nn, q++, psi = hadn[[q]].psi];
      ];
     ][[1]], " sec"];

   Print["Doing measurements took ", Timing[
      measure[psi, mm];
     ][[1]], " sec"];
  ];

(* For reference print table of asymptotically optimum number of iterations *)
Prepend[Table[{n, N[(Pi / 4) * Sqrt[2^n]]}, {n, 3, 15}],
  {"N", "(pi/4)Sqrt[2^N]"}] // TableForm
```

N	(pi/4)Sqrt[2^N]
3	2.22144
4	3.14159
5	4.44288
6	6.28319
7	8.88577
8	12.5664
9	17.7715
10	25.1327
11	35.5431
12	50.2655
13	71.0861
14	100.531
15	142.172

```
(* PROJECT 3: Carry out Grover-search calculations. *)
(* Also PROJECT 5: Increase N using full matrices,
see how memory usage and computation time increases. *)

(* Set number of qubits and compute matrices that don't depend on jc *)
nn = 8;
mbefore = MemoryInUse[];
Print["Memory in use before computing matrices: ",
  NumberForm[mbefore, DigitBlock → 3], " bytes"];
Print["Computing matrices..."];
Print["   ...took ", Timing[
    hadn = Table[expand1[hadamard, q], {q, nn}];
    (* Compute Hadamard gate for each qubit *)
    jj := comporacle[1]; (* Compute J matrix [Eq 27] *)
   ][[1]], " sec"]; (* J matrix is same as oracle matrix for jc=1 *)
mafter = MemoryInUse[];
mdiff = mafter - mbefore;
Print["Memory in use after computing matrices: ",
  NumberForm[mafter, DigitBlock → 3], " bytes, change: ",
  NumberForm[mdiff, DigitBlock → 3], " bytes"];

(* Carry out the quantum calculation
 (can repeat this step without re-computing matrices in above steps *)
compgrover[5, 12, 100];
```

Memory in use before computing matrices: 79,047,544 bytes

Computing matrices...

...took 3.08882 sec

Memory in use after computing matrices: 83,781,432 bytes, change: 4,733,888 bytes

N = 8, Inital state = |0000 0000⟩

Comptutation: Grover search with correct question = |0000 0100⟩ and 12 iterations

Computing oracle matrix took 0. sec

Performing quantum computation took 0.171601 sec

Measurements:

```
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
|0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
```

Most frequent results: |0000 0100⟩(100) |1111 1111⟩(0) |1111 1110⟩(0) |1111 1101⟩(0)

Doing measurements took 0.0156001 sec

```
(* repeat with less than optimal number of iterations *)
compgrover[5, 8, 100];
```

```
N = 8, Inital state = |0000 0000⟩
Comptutation: Grover search with correct question = |0000 0100⟩ and 8 iterations
Computing oracle matrix took 0. sec
Performing quantum computation took 0.124801 sec
Measurements:
 |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
 |0000 0100⟩ |1000 0101⟩ |0000 0100⟩ |0001 0001⟩ |0000 0100⟩ |0000 0100⟩
 |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0110 0101⟩ |0000 0100⟩ |0000 0100⟩
 |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
 |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
 |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0011 0001⟩ |0000 0100⟩ |0000 0100⟩
 |0000 0100⟩ |0000 0100⟩ |0000 1001⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
 |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0000⟩
 |0000 0100⟩ |0000 0100⟩ |1010 1001⟩ |0000 0100⟩ |1101 1110⟩ |0000 0100⟩
 |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0010 0110⟩
 |1101 0101⟩ |0101 1010⟩ |0000 0100⟩ |0001 0000⟩ |0000 0100⟩ |0000 0100⟩
 |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |1010 1011⟩ |0111 1100⟩ |1101 1100⟩
 |0000 0100⟩ |1110 0100⟩ |0000 0100⟩ |0000 0100⟩ |1000 0110⟩ |0000 0100⟩
 |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
 |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0111 1010⟩ |0000 0100⟩ |0000 0100⟩
 |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |1100 0001⟩ |0000 0100⟩
 |0000 0100⟩ |0000 0100⟩ |0000 0100⟩ |0000 0100⟩
Most frequent results: |0000 0100⟩(81) |1110 0100⟩(1) |1101 1110⟩(1) |1101 1100⟩(1)
Doing measurements took 0.0312002 sec
```

```
(* FOR PROJECT 4: Generate full matrix for CNOT gate *)

(* Function takes u, the 2x2 matrix for a one-qubit gate,
and returns cu, the 4x4 matrix for a controlled
 gate with the first bit the controlling bit [Eq 30] *)
makecu[u_] := {{1, 0, 0, 0}, {0, 1, 0, 0},
   {0, 0, u[[1, 1]], u[[1, 2]]}, {0, 0, u[[2, 1]], u[[2, 2]]}};

(* One-qubit not matrix *)
not = {{0, 1}, {1, 0}};

(* Compute CNOT matrix *)
cnot = makecu[not];

(* Function takes matrix for a two-bit gate g2 and qubit numbers q1,q2;
```

```
returns matrix g2 applied to qubits q1,q2 in an nn-qubit system [Eq 37] *)
expand2[g2_, q1_, q2_] := Module[{dd, matq, j, jbits, jq1, jq2, k, kbits, kq1, kq2},
   dd = 2^nn;
   matq = Table[0, {dd}, {dd}]; (* start with matrix all zeroes *)
   For[j = 1, j ≤ dd, j++,
    jbits = IntegerDigits[j - 1, 2, nn]; (* binary digits for j *)
    jq1 = jbits[[q1]];
    jq2 = jbits[[q2]];
    (* bits of j that will index the two-qubit matrix *)
    For[k = 1, k ≤ dd, k++,
     kbits = IntegerDigits[k - 1, 2, nn]; (* binary digits for k *)
     kq1 = kbits[[q1]];
     kq2 = kbits[[q2]];
     (* bits of k that will index the one-qubit matrix *)
     kbits[[q1]] = jbits[[q1]];
     kbits[[q2]] = jbits[[q2]];
     (* these prevent kronecker enforement for bits q1,q2 *)
     If[jbits == kbits, (* this enforces the kronecker deltas *)
      matq[[j, k]] = g2[[2 * jq1 + jq2 + 1, 2 * kq1 + kq2 + 1]];
     ]
    ];
   ];
   matq
  ];


(* Set N=3 and create all needed matrices. Print out 2-qubit and 3-
  qubit CNOT matrices to check against article [Eqs 29,31,32] *)
nn = 3;
had1 = expand1[hadamard, 1];
had2 = expand1[hadamard, 2];
had3 = expand1[hadamard, 3];
pi23 = expand1[phaseshift[Pi / 2], 3];
Print["CNOT = ", cnot // MatrixForm]
c23 = expand2[cnot, 2, 3];
Print["C23 = ", c23 // MatrixForm];
c21 = expand2[cnot, 2, 1];
Print["C21 = ", c21 // MatrixForm];
```

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$C23 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$C21 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

```
(* PROJECT 4a [Fig 6(a)]: Create correlations between 2 and 3 *)
mm = 100;(* number of measurements to do *)

Print["N = ", nn, ", Inital state = ", bstring[1]];
Print["Computation: Had 2; CNOT 2 cont 3"];
dd = 2^nn;
psi = Table[0, {dd}];
psi[[1]] = 1;
psi = had2.psi;
psi = c23.psi;
measure[psi, mm];
```

```
N = 3, Inital state = |000⟩

Computation: Had 2; CNOT 2 cont 3

Measurements:
 |000⟩ |011⟩ |000⟩ |011⟩ |000⟩ |011⟩ |000⟩ |011⟩ |000⟩ |000⟩ |011⟩ |011⟩ |011⟩
 |000⟩ |000⟩ |000⟩ |011⟩ |000⟩ |011⟩ |000⟩ |000⟩ |000⟩ |000⟩ |011⟩ |011⟩ |000⟩
 |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |011⟩ |011⟩ |000⟩ |000⟩ |011⟩ |011⟩ |000⟩ |000⟩
 |011⟩ |000⟩ |011⟩ |000⟩ |011⟩ |000⟩ |000⟩ |011⟩ |000⟩ |011⟩ |011⟩ |011⟩ |000⟩
 |011⟩ |011⟩ |000⟩ |000⟩ |011⟩ |000⟩ |000⟩ |011⟩ |000⟩ |000⟩ |011⟩ |011⟩ |000⟩
 |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |011⟩ |011⟩ |000⟩ |000⟩ |000⟩ |011⟩ |000⟩ |011⟩
 |011⟩ |011⟩ |011⟩ |000⟩ |011⟩ |011⟩ |011⟩ |011⟩ |000⟩ |011⟩ |011⟩ |011⟩ |000⟩
 |011⟩ |011⟩ |000⟩ |011⟩ |000⟩ |011⟩ |011⟩ |000⟩ |011⟩

Most frequent results: |000⟩(52) |011⟩(48) |111⟩(0) |110⟩(0)
```

```
(* PROJECT 4b [Fig 6(b)]: Create cat state *)
mm = 100;(* number of measurements to do *)

Print["N = ", nn, ", Inital state = ", bstring[1]];
Print["Computation: Had 2; CNOT 2 cont 3; CNOT 2 cont 1"];
dd = 2^nn;
psi = Table[0, {dd}];
psi[[1]] = 1;
psi = had2.psi;
psi = c23.psi;
psi = c21.psi;
measure[psi, mm];
```

```
N = 3, Inital state = |000⟩

Computation: Had 2; CNOT 2 cont 3; CNOT 2 cont 1

Measurements:
 |111⟩ |000⟩ |111⟩ |000⟩ |000⟩ |111⟩ |111⟩ |000⟩ |111⟩ |000⟩ |000⟩ |111⟩ |000⟩
 |111⟩ |111⟩ |111⟩ |000⟩ |000⟩ |000⟩ |111⟩ |000⟩ |111⟩ |111⟩ |000⟩ |000⟩ |000⟩
 |111⟩ |111⟩ |000⟩ |000⟩ |000⟩ |000⟩ |111⟩ |111⟩ |111⟩ |000⟩ |000⟩ |000⟩ |111⟩
 |111⟩ |000⟩ |111⟩ |111⟩ |111⟩ |111⟩ |000⟩ |111⟩ |000⟩ |000⟩ |111⟩ |000⟩ |000⟩
 |111⟩ |000⟩ |111⟩ |000⟩ |000⟩ |111⟩ |111⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |111⟩
 |111⟩ |111⟩ |111⟩ |000⟩ |111⟩ |111⟩ |000⟩ |111⟩ |000⟩ |111⟩ |111⟩ |111⟩ |000⟩
 |111⟩ |111⟩ |000⟩ |111⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |111⟩ |111⟩ |111⟩
 |111⟩ |000⟩ |000⟩ |111⟩ |000⟩ |000⟩ |000⟩ |000⟩ |111⟩

Most frequent results: |000⟩(52) |111⟩(48) |110⟩(0) |101⟩(0)
```

```
(* PROJECT 4c [Fig 6(c)]: Unobserved superposition *)
mm = 100;(* number of measurements to do *)

Print["N = ", nn, ", Inital state = ", bstring[1]];
Print["Computation: Had 2; Had 2"];
dd = 2^nn;
psi = Table[0, {dd}];
psi[[1]] = 1;
psi = had2.psi;
psi = had2.psi;
measure[psi, mm];
```

```
N = 3, Inital state = |000⟩

Computation: Had 2; Had 2

Measurements:
 |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩
 |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩
 |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩
 |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩
 |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩
 |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩
 |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩
 |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩ |000⟩

Most frequent results: |000⟩(100) |111⟩(0) |110⟩(0) |101⟩(0)
```

```
(* PROJECT 4d [Fig 6{d}]: Observed superposition *)
mm = 100;(* number of measurements to do *)

Print["N = ", nn, ", Inital state = ", bstring[1]];
Print["Computation: Had 2; CNOT 2 cont 3; Had 2"];
dd = 2^nn;
psi = Table[0, {dd}];
psi[[1]] = 1;
psi = had2.psi;
psi = c23.psi;
psi = had2.psi;
measure[psi, mm];
```

```
N = 3, Inital state = |000⟩

Computation: Had 2; CNOT 2 cont 3; Had 2

Measurements:
 |011⟩ |011⟩ |010⟩ |001⟩ |011⟩ |000⟩ |011⟩ |010⟩ |010⟩ |010⟩ |011⟩ |011⟩ |001⟩
 |000⟩ |001⟩ |010⟩ |010⟩ |010⟩ |000⟩ |001⟩ |000⟩ |001⟩ |010⟩ |011⟩ |011⟩ |000⟩
 |000⟩ |001⟩ |011⟩ |000⟩ |000⟩ |000⟩ |011⟩ |000⟩ |011⟩ |000⟩ |011⟩ |011⟩ |000⟩
 |001⟩ |001⟩ |000⟩ |000⟩ |011⟩ |001⟩ |001⟩ |000⟩ |010⟩ |011⟩ |000⟩ |001⟩ |001⟩
 |011⟩ |010⟩ |011⟩ |010⟩ |001⟩ |011⟩ |010⟩ |001⟩ |001⟩ |001⟩ |010⟩ |010⟩ |010⟩
 |010⟩ |011⟩ |011⟩ |011⟩ |001⟩ |000⟩ |001⟩ |001⟩ |011⟩ |011⟩ |001⟩ |011⟩ |001⟩
 |000⟩ |011⟩ |010⟩ |000⟩ |001⟩ |011⟩ |000⟩ |001⟩ |000⟩ |011⟩ |001⟩ |011⟩ |001⟩
 |000⟩ |010⟩ |001⟩ |010⟩ |000⟩ |010⟩ |010⟩ |010⟩ |010⟩

Most frequent results: |011⟩(28) |001⟩(26) |010⟩(23) |000⟩(23)
```

```
(* FOR PROJECT 6: Sparse matrix functions *)
```

```
(*  Note: Mathematica has built in sparse matrix functions which
  would probably execute much faster but they are not used here.

   Sparse data structures used here:
 (i) Diagonal matrices are simply
   represented as vectors with 2^N complex-number elements.
    (ii) Matrices for one- and two-
  qubit gates are represented as vectors with 2^N elements,
one for each row.  Each element is a list of the form {{e,c},{e,c}...}
 where e is the complex matrix element that goes in column c.
*)

(* Function takes matrix for a one-bit gate g1 and qubit number q;
returns matrix g1 applied to qubit q in an nn-
 qubit system as a sparse matrix. *)
expand1sparse[g1_, q_] := Module[{dd, matg, j, jbits, jq, k1, k2},
   dd = 2^nn;
   matg = {}; (* start with empty list for output vector *)
   For[j = 1, j ≤ dd, j++, (* j is row index *)
    jbits = IntegerDigits[j - 1, 2, nn]; (* binary digits for j *)
    jq = jbits[[q]];
    (* bit of j that will index row of the one-qubit matrix *)
    jbits[[q]] = 0; (* change bit q to 0...*)
    k1 = FromDigits[jbits, 2] + 1;
    (* ...to compute first column index *)
    jbits[[q]] = 1; (* change bit q to 1...*)
    k2 = FromDigits[jbits, 2] + 1;
    (* ...to compute second column index *)
    AppendTo[matg, {{g1[[jq + 1, 1]], k1}, {g1[[jq + 1, 2]], k2}}];
   ];
   matg
  ];

(* Function takes matrix for a two-bit gate g2 and qubit numbers q1,
q2; returns matrix g2 applied to qubits q1,
q2 in an nn-qubit system as a sparse matrix*)
expand2sparse[g2_, q1_, q2_] :=
  Module[{dd, matg, j, jbits, jq1, jq2, g2row, k1, k2, k3, k4},
   dd = 2^nn;
   matg = {}; (* start with empty list for output vector *)
   For[j = 1, j ≤ dd, j++, (* j is row index *)
    jbits = IntegerDigits[j - 1, 2, nn]; (* binary digits for j *)
    jq1 = jbits[[q1]];
```

```
    jq2 = jbits[[q2]];
    (* bits of j that will index the two-qubit matrix *)
    g2row = 2 * jq1 + jq2 + 1;
    (* row of 2-qubit matrix to get elements from *)
    jbits[[q1]] = 0;
    jbits[[q2]] = 0; (* change bits q1,q2 to 0,0...*)
    k1 = FromDigits[jbits, 2] + 1;
    (* ...to compute first column index *)
    jbits[[q1]] = 0;
    jbits[[q2]] = 1; (* change bits q1,q2 to 0,1...*)
    k2 = FromDigits[jbits, 2] + 1;
    (* ...to compute second column index *)
    jbits[[q1]] = 1;
    jbits[[q2]] = 0; (* change bits q1,q2 to 1,0...*)
    k3 = FromDigits[jbits, 2] + 1;
    (* ...to compute third column index *)
    jbits[[q1]] = 1;
    jbits[[q2]] = 1; (* change bits q1,q2 to 1,1...*)
    k4 = FromDigits[jbits, 2] + 1;
    (* ...to compute fourth column index *)
    AppendTo[matg, {{g2[[g2row, 1]], k1},
      {g2[[g2row, 2]], k2},
      {g2[[g2row, 3]], k3},
      {g2[[g2row, 4]], k4}}];
   ];
   matg
  ];


(* Function multiplies sparse matrix matg by state vector psi,
returns result.  This version works for sparse
 matrix with arbitary number of nonzero elements per row. *)
sparsemult[matg_, psi_] := Module[{dd, psiout, j, matgrow, ki, matge, matgk},
   dd = 2^nn;
   psiout = Table[0, {dd}];
   (* initialize output vector to all zeros *)
   For[j = 1, j ≤ dd, j++,
    matgrow = matg[[j]]; (* extract jth row of matg *)
    For[ki = 1, ki ≤ Length[matgrow], ki++,
     matge = matgrow[[ki, 1]];
     matgk = matgrow[[ki, 2]];
     (* extract matrix element and column index *)
     psiout[[j]] += matge * psi[[matgk]]
```

```
    ];
   ];
   psiout
  ];
```

```
(* Function multiplies sparse matrix matg by state vector psi,
returns result.  This version only works for sparse matrices with exactly two
 nonzero elements per row, and is about twice as fast as sparsemult. *)
sparsemult2[matg_, psi_] := Module[{dd, psiout, j, matgrow},
   dd = 2^nn;
   psiout = Table[0, {dd}];
   (* initialize output vector to all zeros *)
   For[j = 1, j ≤ dd, j++,
    matgrow = matg[[j]]; (* extract jth row of matg *)
    psiout[[j]] += matgrow[[1, 1]] * psi[[matgrow[[1, 2]]]] +
      matgrow[[2, 1]] * psi[[matgrow[[2, 2]]]];
   ];
   psiout
  ];
```

```
(* Check sparse representations are correct by printing out some things for N=
 3 [Eqs 20, 31, 38] *)
nn = 3;
h1 = expand1sparse[hadamard, 1];
Print["H1 = ", h1 // MatrixForm];
c23 = expand2sparse[cnot, 2, 3];
Print["C23 = ", c23 // MatrixForm];
```

```
H1 =
⎛ ⎛0.707107⎞  ⎛0.707107⎞ ⎞
⎜ ⎜   1    ⎟  ⎜   5    ⎟ ⎟
⎜ ⎜0.707107⎟  ⎜0.707107⎟ ⎟
⎜ ⎜   2    ⎟  ⎜   6    ⎟ ⎟
⎜ ⎜0.707107⎟  ⎜0.707107⎟ ⎟
⎜ ⎜   3    ⎟  ⎜   7    ⎟ ⎟
⎜ ⎜0.707107⎟  ⎜0.707107⎟ ⎟
⎜ ⎜   4    ⎟  ⎜   8    ⎟ ⎟
⎜ ⎜0.707107⎟  ⎜-0.707107⎟ ⎟
⎜ ⎜   1    ⎟  ⎜   5    ⎟ ⎟
⎜ ⎜0.707107⎟  ⎜-0.707107⎟ ⎟
⎜ ⎜   2    ⎟  ⎜   6    ⎟ ⎟
⎜ ⎜0.707107⎟  ⎜-0.707107⎟ ⎟
⎜ ⎜   3    ⎟  ⎜   7    ⎟ ⎟
⎜ ⎜0.707107⎟  ⎜-0.707107⎟ ⎟
⎝ ⎝   4    ⎠  ⎝   8    ⎠ ⎠
```

$$C23 = \begin{pmatrix} \begin{pmatrix}1\\1\end{pmatrix} & \begin{pmatrix}0\\2\end{pmatrix} & \begin{pmatrix}0\\3\end{pmatrix} & \begin{pmatrix}0\\4\end{pmatrix} \\ \begin{pmatrix}0\\1\end{pmatrix} & \begin{pmatrix}1\\2\end{pmatrix} & \begin{pmatrix}0\\3\end{pmatrix} & \begin{pmatrix}0\\4\end{pmatrix} \\ \begin{pmatrix}0\\1\end{pmatrix} & \begin{pmatrix}0\\2\end{pmatrix} & \begin{pmatrix}0\\3\end{pmatrix} & \begin{pmatrix}1\\4\end{pmatrix} \\ \begin{pmatrix}0\\1\end{pmatrix} & \begin{pmatrix}0\\2\end{pmatrix} & \begin{pmatrix}1\\3\end{pmatrix} & \begin{pmatrix}0\\4\end{pmatrix} \\ \begin{pmatrix}1\\5\end{pmatrix} & \begin{pmatrix}0\\6\end{pmatrix} & \begin{pmatrix}0\\7\end{pmatrix} & \begin{pmatrix}0\\8\end{pmatrix} \\ \begin{pmatrix}0\\5\end{pmatrix} & \begin{pmatrix}1\\6\end{pmatrix} & \begin{pmatrix}0\\7\end{pmatrix} & \begin{pmatrix}0\\8\end{pmatrix} \\ \begin{pmatrix}0\\5\end{pmatrix} & \begin{pmatrix}0\\6\end{pmatrix} & \begin{pmatrix}0\\7\end{pmatrix} & \begin{pmatrix}1\\8\end{pmatrix} \\ \begin{pmatrix}0\\5\end{pmatrix} & \begin{pmatrix}0\\6\end{pmatrix} & \begin{pmatrix}1\\7\end{pmatrix} & \begin{pmatrix}0\\8\end{pmatrix} \end{pmatrix}$$

```
(* PROJECT 6: Grover search with sparse matrices - definitions,
etc prior to actual calculations *)

(* Function computes Grover oracle matrix for correct question =
 jc as a diagonal sparse matrix [Eq 26] *)
comporaclesparse[jc_] := Module[{dd, matd, j},
   dd = 2^nn;
   matd = Table[1, {dd}]; (* start with column vector all ones *)
   matd[[jc]] = -1; (* diagonal element jc,jc is -1 *)
   matd
  ];

(* Function carries out Grover quantum calculation
 for correct answer jc and iterations its [Fig 4] *)
compgroversparse[jc_, its_, mm_] := Module[{oracle, dd, psi, q, it},
   Print["N = ", nn, ", Inital state = ", bstring[1]];
   Print["Comptutation: Grover search with correct question = ",
    bstring[jc], " and ", its, " iterations"];
   Print["Computing oracle matrix took ", Timing[
      oraclesparse = comporaclesparse[jc];
     ][[1]], " sec"];

   Print["Performing quantum computation took ", Timing[
      (* set initial state = |000> *)
      dd = 2^nn;
      psi = Table[0, {dd}];
      psi[[1]] = 1;

      (* perform intial Hadamard on each qubit *)
      For[q = 1, q ≤ nn, q++, psi = sparsemult2[hadnsparse[[q]], psi]];

      (* do Grover steps *)
```

```
      For[it = 1, it ≤ its, it++,
       psi = oraclesparse * psi;
       For[q = 1, q ≤ nn, q++, psi = sparsemult2[hadnsparse[[q]], psi]];
       psi = jjsparse * psi;
       For[q = 1, q ≤ nn, q++, psi = sparsemult2[hadnsparse[[q]], psi]];
      ];
     ][[1]], " sec"];
   Print["Doing measurements took ", Timing[
      measure[psi, mm];
     ][[1]], " sec"];
  ];

(* For reference print table of asymptotically optimum number of iterations *)
Prepend[Table[{n, N[(Pi / 4) * Sqrt[2^n]]}, {n, 10, 25}],
  {"N", "(pi/4)Sqrt[2^N]"}] // TableForm
```

N	(pi/4)Sqrt[2^N]
10	25.1327
11	35.5431
12	50.2655
13	71.0861
14	100.531
15	142.172
16	201.062
17	284.345
18	402.124
19	568.689
20	804.248
21	1137.38
22	1608.5
23	2274.76
24	3216.99
25	4549.51

```
(* PROJECT 6: Carry out Grover-search calculations using sparse matrices *)
(* Set number of qubits and compute matrices that don't depend on jc *)
nn = 10;
mbefore = MemoryInUse[];
Print["Memory in use before computing matrices: ",
  NumberForm[mbefore, DigitBlock → 3], " bytes"];
Print["Computing matrices..."];
Print["   ...took ", Timing[
    hadnsparse = Table[expand1sparse[hadamard, q], {q, nn}];
    (* Hadamard for each qubit *)
    jjsparse := comporaclesparse[1]; (* J matrix [Eq 27] *)
   ][[1]], " sec"];
mafter = MemoryInUse[];
mdiff = mafter - mbefore;
Print["Memory in use after computing matrices: ",
  NumberForm[mafter, DigitBlock → 3], " bytes, change: ",
  NumberForm[mdiff, DigitBlock → 3], " bytes"];

(* Carry out the quantum calculation
 (can repeat this step without re-computing matrices in above steps *)
compgroversparse[34, 25, 100];
```

```
Memory in use before computing matrices: 87,115,408 bytes

Computing matrices...

   ...took 0.156001 sec

Memory in use after computing matrices: 88,926,424 bytes, change: 1,811,016 bytes

N = 10, Inital state = |00 0000 0000〉

Comptutation: Grover search with correct question = |00 0010 0001〉 and 25 iterations

Computing oracle matrix took 0. sec

Performing quantum computation took 4.36803 sec

Measurements:
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉
 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉 |00 0010 0001〉

Most frequent results: |00 0010 0001〉(100)
  |11 1111 1111〉(0) |11 1111 1110〉(0) |11 1111 1101〉(0)

Doing measurements took 0.0312002 sec
```

```
(* END OF PROGRAM "2015-XX-XX Quantum computing.nb" *)
```